electronics

MDPI

Article

A Reinforcement Learning Method of Solving Markov Decision Processes: An Adaptive Exploration Model Based on Temporal Difference Error

Xianjia Wang [1,2], Zhipeng Yang [1,3,*], Guici Chen [1,3] and Yanli Liu [1,3]

[1] Hubei Province Key Laboratory of Systems Science in Metallurgical Process, Wuhan University of Science and Technology, Wuhan 430065, China; wangxj@whu.edu.cn (X.W.); chenguici@wust.edu.cn (G.C.); yanlil2008@wust.edu.cn (Y.L.)
[2] Economics and Management School, Wuhan University, Wuhan 430072, China
[3] College of Science, Wuhan University of Science and Technology, Wuhan 430065, China
* Correspondence: yangzhipeng@wust.edu.cn

Abstract: Traditional backward recursion methods face a fundamental challenge in solving Markov Decision Processes (MDP), where there exists a contradiction between the need for knowledge of optimal expected payoffs and the inability to acquire such knowledge during the decision-making process. To address this challenge and strike a reasonable balance between exploration and exploitation in the decision process, this paper proposes a novel model known as Temporal Error-based Adaptive Exploration (TEAE). Leveraging reinforcement learning techniques, TEAE overcomes the limitations of traditional MDP solving methods. TEAE exhibits dynamic adjustment of exploration probabilities based on the agent's performance, on the one hand. On the other hand, TEAE approximates the optimal expected payoff function for subprocesses after specific states and times by integrating deep convolutional neural networks to minimize the temporal difference error between the dual networks. Furthermore, the paper extends TEAE to DQN-PER and DDQN-PER methods, resulting in DQN-PER-TEAE and DDQN-PER-TEAE variants, which not only demonstrate the generality and compatibility of the TEAE model with existing reinforcement learning techniques but also validate the practicality and applicability of the proposed approach in a broader MDP reinforcement learning context. To further validate the effectiveness of TEAE, the paper conducts a comprehensive evaluation using multiple metrics, compares its performance with other MDP reinforcement learning methods, and conducts case studies. Ultimately, simulation results and case analyses consistently indicate that TEAE exhibits higher efficiency, highlighting its potential in driving advancements in the field.

Keywords: Markov decision process; reinforcement learning; exploration and exploitation; adaptive; decision making

Citation: Wang, X.; Yang, Z.; Chen, G.; Liu, Y. A Reinforcement Learning Method of Solving Markov Decision Processes: An Adaptive Exploration Model Based on Temporal Difference Error. *Electronics* **2023**, *12*, 4176. https://doi.org/10.3390/electronics12194176

Academic Editor: Andrea Asperti

Received: 13 September 2023
Revised: 30 September 2023
Accepted: 4 October 2023
Published: 8 October 2023

1. Introduction

In practice, there are various multi-stage decision problems involving uncertainty. This uncertainty is often characterized by Markov processes, known as the Markov decision processes (MDP) [1–4]. The backward recursive method, based on the optimality principle, is a common approach for solving MDP [5–7]. However, this approach assumes that the optimal expected payoff of subsequent sub-processes is known when making decisions at any state and any point in the decision process. In reality, the optimal expected payoff of subsequent sub-processes is not fully known during the decision process, creating a contradiction between the traditional MDP decision method and the conditions present in actual decision making. Nevertheless, despite lacking complete knowledge of the optimal payoff of future sub-processes, decision makers can approximate their future payoffs by learning from the process's history. This enables them to make informed decisions in the presence of uncertainty.

Reinforcement Learning (RL) techniques such as Deep Q-Network (DQN) [8], Double Deep Q-Network (DDQN) [9], and Soft Actor-Critic (SAC) [10] effectively solve MDP with unknown or uncertain dynamics. These methods estimate expected payoffs iteratively based on observed outcomes and update decision strategies accordingly [11–13]. They are commonly referred to as RL method for MDP (RLMDP) [14,15]. Tasks with Markovian properties, such as scheduling unmanned drones for emergency power supply transport or formulating distribution strategies for warehouse locations in the supply chain, can all be addressed using the RLMDP method [16,17]. RLMDP primarily relies on learning from empirical knowledge, allowing decision makers to adapt their strategies over time to maximize their payoffs [18–20]. However, it is crucial to acknowledge that incorporating learning into decision making introduces additional complexities [21,22]. One complexity is the trade-off between exploration and exploitation, where decision makers must balance between exploring new possibilities and exploiting their current knowledge.

In recent years, researchers have proposed various methods to address the exploration–exploitation trade-off in RLMDP approaches [23–26]. However, one limitation of many of these methods is that they often require manual adjustment of parameters, which can make them less effective and limit their ability to generalize well. Additionally, while RLMDP offers resource efficiency advantages, it struggles to efficiently utilize all empirical data [27–29]. To overcome this limitation, the Prioritized Experience Replay (PER) sampling technique has been introduced and integrated with DQN and DDQN methods, resulting in DQN-PER and DDQN-PER approaches [8,9,30]. By incorporating the PER technique, DQN and DDQN methods selectively learn from empirical data, significantly improving decision-making efficiency. Furthermore, in reference [31], the computational methods for balancing exploration and exploitation in RL are investigated, starting from bio-inspired neural networks. Literature [32] utilizes a substantial amount of expert data to pre-train the transformer decoder, guiding the agent's behavior in the early learning stages. After reaching a certain learning threshold, an epsilon-greedy strategy is employed to determine actions. However, these methods currently lack the ability to dynamically update the exploration probability based on agent–environment interactions, potentially leading to suboptimal or unstable solutions [33].

This paper proposes the Temporal difference Error-based Adaptive Exploration (TEAE) model, which aims to balance exploration and exploitation in the decision-making process and optimize the expected payoff of the subsequent sub-processes. TEAE is an adaptive $\epsilon(s_t)$-greedy exploration model that utilizes the Temporal Difference (TD) Error between dual networks to quantify exploration probabilities. It enables agents to dynamically adjust exploration probability based on their knowledge, leveraging the magnitude of TD error for informed decision making. In the initial learning stages, when the agent's understanding of the environment is uncertain and TD error is high, TEAE encourages frequent exploration. As knowledge increases and TD error decreases, exploration opportunities are reduced. Subsequently, the TEAE model combines with two existing methods, DQN-PEF and DDQN-PER, resulting in two new approaches: DQN-PER-TEAE and DDQN-PER-TEAE. These approaches facilitate the attainment of optimal solutions and improve performance in uncertain Markov game environments. Finally, parameter sensitivity validation, performance comparison experiments and the case study validate the effectiveness of the TEAE approach compared to existing methods for solving MDP. The code for the method proposed in the paper can be found at https://github.com/zhipeng-yang/RLMDP-TEAE.git/, accessed on 1 October 2023.

Compared to the state-of-the-art literature, this paper makes the following contributions:

- In the dual-network RL algorithm, we introduce an TEAE model to address the trade-off between exploration and exploitation. This model dynamically adjusts the exploration probability, effectively balancing the exploration of new possibilities and the exploitation of existing knowledge. As a result, it can maximize the optimal expected payoff of the posterior subprocess.

- We have developed two RL algorithms, namely DQN-PER-TEAE and DDQN-PER-TEAE, which are built upon the TEAE model we proposed. These algorithms incorporate the adaptive exploration mechanism of the TEAE model, enabling the agent to dynamically adjust its exploration strategy based on its current knowledge. This capability allows the agent to effectively explore unfamiliar or complex states, thereby enhancing the efficiency and effectiveness of learning in solving MDP.
- We evaluate the performance of our proposed model and algorithm using various evaluation criteria and cases. Through comparisons with state-of-the-art methods, we observe that our TEAE method outperforms existing approaches in terms of efficiency for solving MDP.

The rest of this study is arranged as follows. In the following section, we provid a notation explanation and formalize the problem under study. Section 3 first describes the overall framework flow of the TEAE model, then explains the key works in the TEAE model, and finally introduces the DQN-PER-TEAE and DDQN-PER-TEAE algorithms. Section 4 introduces experimental evaluation metrics, experimental results, and case analysis. In the last section, we summarize the main conclusions and indicate future research directions.

2. Notation and Problem Formalization

Table 1 describes all mathematical symbols used in this paper.

Table 1. Notation.

Symbol	Description
S	State space.
s_t	The state at time t.
A	Action space.
$\|A\|$	The size of the action space.
a_t	The actions executed at time t.
P	Probability of state transfer.
R	Reward function.
r_t	Reward obtained at time t.
γ	Discount factor.
$\mathbb{E}$	Expectation function.
Q	Action value function.
Q^π	Action value function under strategy π.
Q^*	Optimal action value function.
π	The set of strategies.
π^*	Optimal strategy.
$s \sim p(\cdot)$	The state s follows a probability distribution p.
α	Learning rate.
$\delta(t)$	TD error at time t.
ε	Parameters of the ϵ-greedy algorithm.
τ	Parameters of the SoftMax algorithm.
ω'	Parameters of the target network in the DQN algorithm.
ω	Parameters of the current network in the DQN algorithm.
θ'	Parameters of the target network in the DDQN algorithm.
θ	Parameters of the current network in the DDQN algorithm.
σ	Parameters of the TEAE model, inverse sensitivity coefficient.
β	Parameters in the TEAE model, usually taken as $\frac{1}{\|A\|}$.
D	Experience replay pool.
C	The size of the experience replay pool.
N	Number of iteration epochs.
n	Number of iteration time steps in the epoch.
F	Frequency of updating parameters of the target network.

MDP is an analytical framework used to study multi-stage decision-making problems with Markov chain properties. It is also a mathematical framework used for simulating decision problems in RL. MDP represents the interaction between an agent and its environ-

ment in a continuous manner, where the agent takes actions that transition between states and receives rewards based on its actions [34].

Definition 1. *A finite-stage MDP can be represented as a six-tuple* (T, S, A, P, R, γ)*, in which*

- T *represents a finite time period, with* $t = \{0, 1, 2, ..., T\}$.
- S *indicates the state space. Assuming that in each time period* t, $S_t = S$*, meaning that the state space remains the same in each time period, we have* $S = \{s_1, s_2, \ldots, s_m\}$*, where* $s_t \in S$*. In this study, the state mainly uses mathematical symbols to record information such as the location, instantaneous velocity, and energy value of the agent in the simulator in real time.*
- A *denotes the action space. Assuming that in each time period* t, $A_t = A$*, meaning that the action space remains consistent in each time period, we have* $A = \{a_1, a_2, \ldots, a_k\}$*, where* $a_t \in A$*. In this research, the action is mainly the agent moving up, down, left, right or shooting.*
- P *is the state transfer probability function. At time* t*, when state* s_t *takes action* a_t*, it transitions to state* s_{t+1} *at time* $t+1$*. The probability of this state transition within the stage is denoted as* $P_{s_t,s_{t+1}}(a_t)$*. It can generally be represented as:* $(t, s_t) \xrightarrow[P_{s_t,s_{t+1}}(a_t)]{a_t \in A_t} (t+1, s_{t+1})$.
- R *is the reward (payoff) function. At time* t*, when state* s_t *takes action* a_t *and transitions to state* s_{t+1} *at time* $t+1$*, the immediate reward obtained in state* s_t *is denoted as* $R(s_t, a_t)$*, and generally simplified as* r_t.
- γ *is the discount factor. It is a value between* 0 *and* 1*, where a higher value places more emphasis on long-term rewards.*

At each time step, the agent observes the current states, selects an action a_t from the set of available actions A, and performs the action in the environment. The environment then transitions to a new state based on the state transition probabilities $P_{s_t,s_{t+1}}(a_t)$. The agent receives an immediate reward r_t for this transition. The goal of the agent is to learn a policy $\pi(a_t|s_t)$ that maps each state s_t to an action a_t, in order to maximize the expected cumulative discounted reward over time. Moreover, the state-action Q function is defined as Equation (1).

$$Q^{\pi}(s_t, a_t) = \mathbb{E}^{\pi}[r_t|s_t, a_t]. \tag{1}$$

An optimal policy can maximize the cumulative discounted reward. A policy π^* is defined to be the optimal policy, if and only if its expected payoff is greater than or equal to that of π for all the state-action pair (s, a).

$$Q^{\pi^*}(s, a) \geq Q^{\pi}(s, a). \tag{2}$$

The Q^{π^*} (in the following, it will be abbreviated as Q^*) function is the ultimate goal of an MDP, as it represents the maximum expected reward achievable by executing a given action in a given state and then following the optimal policy thereafter. The Q^* function is the same for all optimal policies, and it is typically denoted as Q^*. More formally, for a given state s_t and action a_t, the Q^* function can be defined as follows:

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim P_{s_t,s_{t+1}}(a_t)}[R(s_t, a_t) + \gamma \max_{a' \in A} Q^*(s_{t+1}, a')], \tag{3}$$

where $R(s_t, a_t)$ is the immediate reward obtained after executing action a_t in state s_t at time t, s_{t+1} is the next state, a' is any action that can be taken at s_{t+1}, and γ is a discount factor that determines the weight given to future rewards. According to the principle of dynamic programming [35], Equation (3) can be written as follows:

$$Q^*(s_t, a_t) = \max\{R(s_t, a_t) + \gamma \sum_{a' \in A} P_{s_t,s_{t+1}}(a_t) Q^*(s_{t+1}, a')\}. \tag{4}$$

Since the transition probabilities of states ($P_{s_t,s_{t+1}}(a_t)$) and the optimal payoffs of posterior sub-processes ($\max_{a' \in A} Q^*(s_{t+1}, a')$) in Equations (3) and (4) are often unknown, TD methods are commonly used to address prediction problems by leveraging empirical data. When given some experience following a policy π, the TD approach updates the estimate $Q(s_t, a_t)$ for the nonterminal states s_t occurring in that experience. At time $t+1$, the TD approach immediately forms a target and makes a useful update using the observed reward r_{t+1} and the estimate $Q(s_t, a_t)$. The simplest update rule for the TD approach is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)], \tag{5}$$

where α is the learning rate.

It is important to note that the quantity in brackets in Equation (5) represents an error, which quantifies the difference between the estimated value of $Q(s_t, a_t)$ and the better estimate $r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})$. This quantity is commonly referred to as the TD error, and it arises in various forms throughout RL.

$$\delta(t) \doteq r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t). \tag{6}$$

Furthermore, since the Q value is an estimate, there is a risk of the agent falling into local optima if it has not explored all state-action combinations in the past. Therefore, it is essential to not only use experience-based methods but also explore new methods to find the global optimal solution. The main challenge in RL is to strike a balance between exploitation and exploration, i.e., to decide whether to choose actions based on the current knowledge (exploitation) or to explore new actions that may lead to better outcomes (exploration). This paper aims to address this problem by minimizing the difference between the $Q^*(s, a)$ and the current $Q(s_t, a_t)$.

3. TEAE Model

In this section, we first introduce RLMDP method, which is a class of methods that approximate the solution of MDP using RL algorithms. Then, we present the TEAE model proposed in this study. Subsequently, we integrate the TEAE model into two existing algorithms, resulting in the development of two new algorithms named DQN-PER-TEAE and DDQN-PER-TEAE.

3.1. Reinforcement Learning Method for MDP

The conventional approach for solving MDP is the backward recursive method, which assumes knowledge of the optimal expected payoff for all states and at all times during the decision-making process. However, in practical decision-making scenarios, such information is often unknown. This limitation has led to the emergence of RLMDP as an effective method for addressing MDP with unknown or uncertain dynamics. RLMDP provides a framework that enables agents to learn and adapt their behavior without prior knowledge of the underlying MDP dynamics. Instead of relying on known payoff values, agents explore the state-action space, learn from observed rewards, and adjust their policies accordingly. This trial-and-error learning approach allows agent to effectively solve MDP with unknown or uncertain dynamics, where traditional methods based on the backward recursive method may be impractical or infeasible [36–38].

The workflow diagram of the RLMDP is presented in Figure 1. The RLMDP involves two networks: a current network with parameters ω and a target network with parameters ω'. The current network is responsible for selecting actions and updating model parameters based on the observed rewards. On the other hand, the target network is used to calculate the target Q value, which guides the learning process. In order to reduce the correlation between the target Q value and the current Q value, the parameters of the current network are periodically copied to the target network. The critical aspect of the RLMDP lies in

striking a balance between exploration and exploitation, and the proposed TEAE in this paper is precisely designed to address this issue.

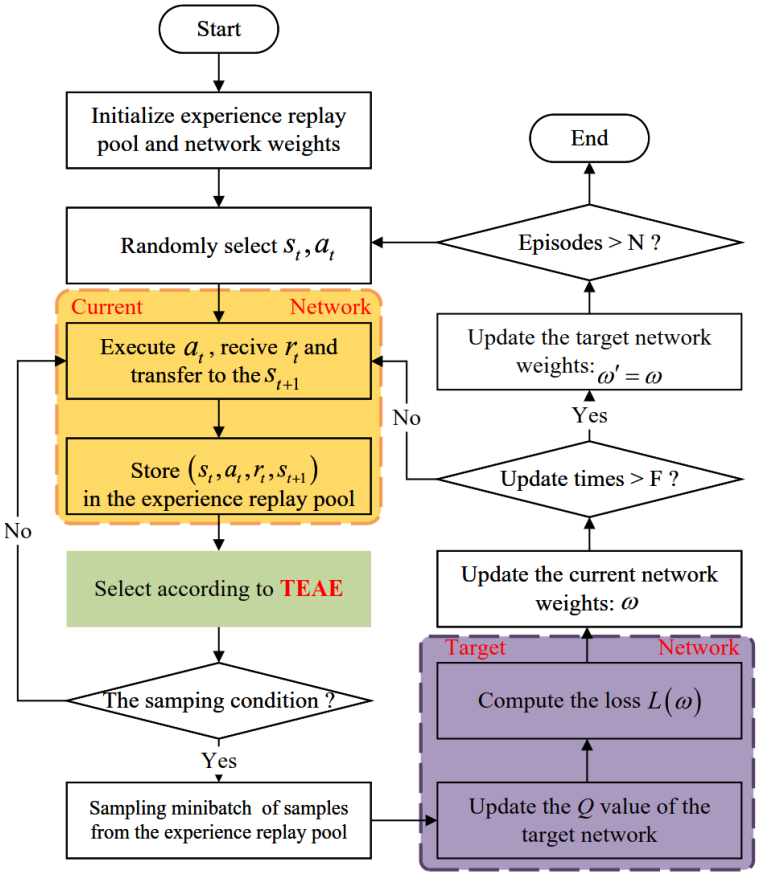


Figure 1. The workflow diagram of the RL methods for solving the MDP.

In more detail, RLMDP starts by randomly selecting an initial state $s_t = s_0$ and an initial action $a_t = a_0$. The agent then executes this action to obtain an immediate reward r_t, and transitions to the next state s_{t+1}. The data (s_t, a_t, r_t, s_{t+1}) is stored in an experience replay pool D. Next, the agent selects the next action a_{t+1} based on the TEAE policy. If the sampling condition is met, the RLMDP samples a batch of data from the experience replay pool D, and then updates the Q value of the target network. The loss function $L(\omega)$ is then computed, representing the difference between the target Q value and the current Q value. The current network parameters ω are updated using gradient descent, as described in Equations (7) and (8). Finally, at intervals of F epochs, the parameters of the current network are periodically copied to the target network.

$$L(\omega) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(Q_{tar}(s', a', \omega') - Q(s, a, \omega)\right)^2\right]. \tag{7}$$

$$\nabla_\omega L(\omega) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[L'(\omega)\right], \tag{8}$$

where $(s, a, r, s') \sim U(D)$ indicates that data (s, a, r, s') is sampled from experience replay pool D, and

$$L'(\omega) = \left(Q_{tar}(s', a', \omega') - Q(s, a, \omega)\right)\nabla_\omega Q(s, a, \omega).$$

In the RLMDP, the action selection process of the current network plays a crucial role in determining the speed of parameter updates. This aspect directly impacts the efficiency of solving MDP tasks. To address this issue, the proposed TEAE model is introduced. TEAE is an adaptive action selection exploration model that dynamically updates the agent's exploration probability in real-time based on the TD error of both the current

and target networks. This approach ensures that the agent continuously learns from the environment by exploring new possibilities while also exploiting its existing knowledge. Compared to existing literature, such as the work by [8], TEAE offers a more effective method of learning from empirical data and avoiding getting stuck in local optima. The TEAE model is further elaborated in the subsequent section, providing a detailed description of its functionality and benefits.

3.2. TD Error-Based Adaptive Exploration Algorithm

The two common algorithms similar to the proposed TEAE model are the ϵ-greedy and the softmax. The ϵ-greedy algorithm suggests that the agent chooses actions randomly with a fixed probability at each time step, instead of greedily selecting an optimal action based on the Q function. When the random number ξ generated by the algorithm is less than the exploration probability ϵ set in the ϵ-greedy algorithm, the current optimal action based on the Q function is selected with a probability of $\frac{1}{|A|}$, where $|A|$ represents the size of the action space. Otherwise, the current optimal action based on the Q function is selected with a probability of 1. The mathematical expression is as follows:

$$p(a^*|s) = \begin{cases} \frac{1}{|A|} & \text{if } \xi < \epsilon, \\ 1 & \text{if } \xi \geq \epsilon. \end{cases} \tag{9}$$

where a^* is the current optimal action about the Q function, $|A|$ is the size of the action space, and $0 \leq \xi \leq 1$ is a uniform random number.

In contrast to the ϵ-greedy algorithm, the softmax algorithm uses the Boltzmann distribution to rank the Q values of the actions and determine the probability of action selection. The mathematical expression for softmax is as follows:

$$p(a_i|s) = \frac{e^{\frac{Q(s,a_i)}{\tau}}}{\sum_{j=1}^{|A|} e^{\frac{Q(s,a_j)}{\tau}}}, \tag{10}$$

where $i = 1, 2, 3, \ldots, |A|$, $p(a_i|s)$ is the probability of selecting action a_i at state s, $Q(s, a_i)$ is the estimated Q value of the action a_i at state s, τ is the temperature parameter, and $|A|$ is the size of the action space. The temperature parameter controls the degree of exploration. When τ is high, actions with lower Q values have a higher probability of being selected, leading to more exploration. When τ is low, actions with higher Q values have a higher probability of being selected, leading to more exploitation.

In practical applications, the ϵ-greedy and softmax methods require fixed parameters to be used, such as ϵ in ϵ-greed and τ in softmax, and these parameters usually need to be set manually. However, these fixed parameters do not take into account the learning process of the agent. This motivates the design of exploration algorithms that can adapt their behavior according to some measure of the agent's learning progress. Therefore, in this paper, an adaptive $\epsilon(s_t)$-greedy exploration algorithm is designed based on TD error in the dual networks, named TEAE. Firstly, we substitute $x = e^{\frac{Q_{tar}(s,a,\omega')}{\sigma}}$ and $y = e^{\frac{Q(s,a,\omega)}{\sigma}}$ into $f(x, y) = |\frac{x}{x+y} - \frac{y}{x+y}|$ and obtain:

$$f(x, y) = \frac{1 - e^{\frac{-|Q_{tar}(s,a,\omega') - Q(s,a,\omega)|}{\sigma}}}{1 + e^{\frac{-|Q_{tar}(s,a,\omega') - Q(s,a,\omega)|}{\sigma}}}, \tag{11}$$

where $Q_{tar}(s, a, \omega')$ is the Q value for the target network fitted state action pair (s, a) and $Q(s, a, \omega)$ is the Q value for the current network fitted state action pair (s, a). ω' and ω are the parameters of the target and current networks, respectively, and σ is a positive

constant called the inverse sensitivity factor. As stated in the previous section, the TD error is generally expressed as $\delta(t)$. Thus, Equation (11) can be abbreviated as:

$$g(\delta(t), \sigma) = \frac{1 - e^{\frac{-|\delta(t)|}{\sigma}}}{1 + e^{\frac{-|\delta(t)|}{\sigma}}}. \tag{12}$$

From Figure 2, it is evident that for a fixed value of σ, the function $g(\delta(t), \sigma)$ is monotonically non-decreasing with respect to $\delta(t)$, and its value lies within the interval $[0, 1)$. Thus, in the extreme scenario where the Q function converges and $\delta(t)$ decreases to 0, the exploration probability will also converge to 0, resulting in the agent choosing actions solely based on greed. However, even in this case, the agent can still make good decisions since the Q function has already converged.

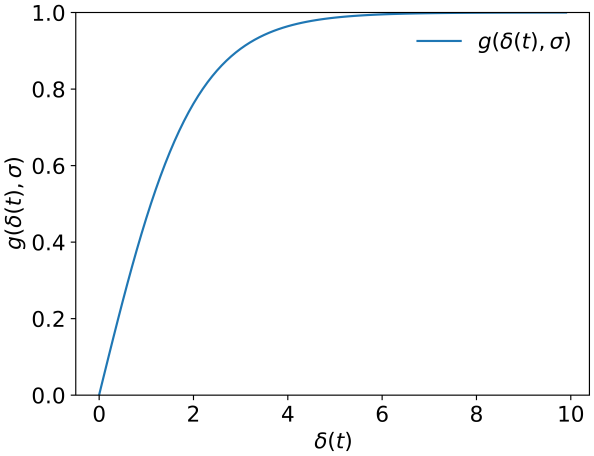


Figure 2. The graph of the function on TD error.

The update process for the exploration probability in TEAE follows a simple rule: as shown in Equation (13), it first provides the exploration probability $\epsilon(s_{t-1})$ for the state s_{t-1} at time step $t-1$. Then, the estimated value of $\epsilon(s_t)$ can be calculated as $(1-\beta) \cdot g(\delta(t), \sigma) + \beta \cdot \epsilon(s_{t-1})$. Here, $\beta \in (0, 1)$, typically set to $\frac{1}{|A|}$, which determines the influence of TD error on the exploration probability, where $|A|$ represents the size of the action space.

$$\epsilon(s_t) = (1-\beta) \cdot g(\delta(t), \sigma) + \beta \cdot \epsilon(s_{t-1}). \tag{13}$$

The TEAE algorithm takes into account the agent's cognition of the environment and the current level of learning. Moreover, the pseudocode is shown in Algorithm 1. It is designed to improve the exploration strategy of an agent in a RL setting. By incorporating the TD error into the softmax algorithm, the exploration probability is adjusted dynamically based on the agent's current knowledge of the environment. The TD error serves as a measure of the discrepancy between the Q values obtained from the current network and the target network. It reflects the uncertainty or confidence of the agent's predictions about the expected rewards when transitioning between states and taking specific actions. By monitoring the TD error, the agent can gauge the accuracy of its predictions and adjust its exploration and exploitation strategies accordingly. A higher TD error indicates greater uncertainty, prompting the agent to explore more to improve its understanding of the environment, while a lower TD error signifies higher confidence, leading the agent to exploit its current knowledge more extensively. This adaptive approach helps to strike a balance between exploration and exploitation, enabling the agent to effectively navigate and learn from the environment over time. The formula Equation (12) defines a function of the TD error based on the expected behavior, which is used to adaptively adjust the

exploration probability $\epsilon(s_t)$ for the current state s_t of the agent. The updated exploration probability is then used in action selection according to Equation (13).

Algorithm 1 TEAE

Initialize $\epsilon(s_0) = 1$
Randomly select $s_0 \in S, a_0 \in A$
for $t = 0$ to n **do**
  Obtain r_t, s_{t+1} after Execute a_t
  $s_t \leftarrow s_{t+1}$
  Update $\delta(t)$ according to the $|Q_{tar} - Q|$
  Update $\epsilon(s_{t+1})$ according to the Equation (13)
  ξ = Random()
  if $\xi < \epsilon(s_{t+1})$ **then**
    Select $a_{t+1} = a^*$ with probability $\frac{1}{|A|}$
  else
    Select $a_{t+1} = a^*$ with probability 1
  end if
end for

The TEAE algorithm starts with initializing the exploration probability $\epsilon(s_0)$ as one for all states, which ensures that the agent can explore the entire action space. During the learning process, the action exploration strategy is defined by Equation (14), where a^* is the current optimal action based on the Q function, $|A|$ is the size of the action space, ξ is a uniform random number between 0 and 1, and $\epsilon(s_t)$ is obtained using Equation (13).

$$p(a^*|s) = \begin{cases} \frac{1}{|A|} & \text{if } \xi < \epsilon(s_t), \\ 1 & \text{if } \xi \geq \epsilon(s_t). \end{cases} \tag{14}$$

Furthermore, the TEAE algorithm updates the TD error based on the difference between the Q values obtained from the current network and the target network, then updates the exploration probability $\epsilon(s_{t+1})$ using Equation (13). Finally, it generates a uniform random number ξ and compares it with $\epsilon(s_{t+1})$ to decide whether to select the current optimal action a^* with probability 1 or with probability $\frac{1}{|A|}$, where $|A|$ is the size of the action space. Next, we will introduce two RL algorithms that are improved based on TEAE.

3.3. Reinforcement Learning Algorithm Based on TEAE

In this section, we introduce two RL algorithms that incorporate the TEAE adaptive exploration mechanism: DQN-PER-TEAE algorithm and DDQN-PER-TEAE algorithm.

(1) DQN-PER-TEAE

The DQN-PER-TEAE algorithm is an extension of the DQN-PER algorithm that incorporates the TEAE model for adaptive exploration. The DQN-PER algorithm uses the PER sampling technique during training and employs a dual-network structure to fit the state-action value function. The current network $Q(s, a, \omega)$ is used to select the action and update the network parameters, while another target network $Q_{tar}(s, a, \omega')$ is used to calculate the target Q value.

The TD error in the DQN-PER-TEAE algorithm, which is defined as the difference between the Q value of the target network in the current state and the current network, is shown in Equation (15).

$$\delta(t) \doteq Q_{tar}(s_t, a_t, \omega') - Q(s_t, a_t, \omega), \tag{15}$$

where ω' and ω are the network parameters of the target network and the current network, respectively. Furthermore, in the DQN-PER-TEAE algorithm, the rule for updating the Q

value of the target network is: if episode terminates at time step $t+1$, then update according to Equation (16), otherwise update according to Equation (17), where a' represents the action candidate set in state s_{t+1}. The formula for calculating the Q value of the current network is shown in Equation (18).

$$Q_{tar}(s_t, a_t, \omega') = r_t. \tag{16}$$

$$Q_{tar}(s_t, a_t, \omega') = r_t + \gamma \max_{a'} Q_{tar}(s_{t+1}, a', \omega'). \tag{17}$$

$$Q(s_t, a_t, \omega) = r_t + \gamma Q(s_{t+1}, a_{t+1}, \omega). \tag{18}$$

Algorithm 2 presents the pseudocode of the DQN-PER-TEAE algorithm. The initialization phase from lines 1 to 4 includes setting the experience replay pool capacity C, network parameters for the current and target networks, and exploration rate $\epsilon(s_0) = 1$. The algorithm proceeds with an iterative loop from line 5, where a random state and action are selected, and the action is executed to obtain an immediate reward r_t, transitioning to the next state s_{t+1}. The transition data (s_t, a_t, r_t, s_{t+1}) is stored in the experience replay pool D. Lines 10 to 13 represent the core of the TEAE model, where the TD error $\delta(t)$ is updated using Equation (15), followed by the update of $\epsilon(s_{t+1})$ using Equation (13). A uniform random number ξ is generated, and a new action is selected using Equation (14). Line 15 implements the PER sampling method to sample small batches of data from the experience replay pool D. The target network Q value is calculated using Equations (16) and (17), and the Adam optimization algorithm is used to optimize the network parameters ω. Finally, the target network parameters are updated to the current network parameters every F epochs.

Algorithm 2 DQN-PER-TEAE

1: Initialize experience pool D to capacity C
2: Initialize Q with random weight ω
3: Initialize Q_{tar} with random weight $\omega' = \omega$
4: Initialize $\epsilon(s_0) = 1$
5: **for** $T = 1$ to N **do**
6: Randomly select $s_0 \in S, a_0 \in A$
7: **for** $t = 1$ to n **do**
8: Obtain r_t, s_{t+1} after Execute a_t
9: Store transition (s_t, a_t, r_t, s_{t+1}) in D
10: Update $\delta(t)$ according to the Equation (15)
11: Update $\epsilon(s_{t+1})$ according to the Equation (13)
12: ξ = Random()
13: Select a_{t+1} according to the Equation (14)
14: **if** The samping condition is satisfied **then**
15: Sample minibatch of transition (s_j, a_j, r_j, s_{j+1}) from D (via PER [30])
16: Update $Q_{tar}(s_j, a_j, \omega')$ according to the Equations (16) and (17)
17: Update $\omega \leftarrow$ Equation (8) (via Adam [39])
18: **end if**
19: **end for**
20: **if** T % $F == 0$ **then**
21: $\omega' \leftarrow \omega$
22: **end if**
23: **end for**

The DQN-PER-TEAE algorithm can dynamically adjust the exploration probability based on the value estimates of state-action pairs by both the current and target networks at each time step. This allows the agent to balance exploration and exploitation based on its perception of the environment. By doing so, the algorithm can improve the overall

robustness of the DQN-PER algorithm and makes the collected empirical data more efficient and useful.

(2) DDQN-PER-TEAE

The DDQN-PER-TEAE algorithm is an extension of the DDQN-PER algorithm that incorporates the TEAE model for adaptive exploration. Similar to the DDQN-PER algorithm, the DDQN-PER-TEAE algorithm uses the PER sampling technique during training and employs a dual-network structure to fit the state-action value function. However, to address the overestimation problem in the DQN-PER algorithm, the DDQN-PER algorithm decouples the action selection of the target Q value and the calculation of the target Q value.

In the DDQN-PER-TEAE algorithm, θ and θ' represent the parameters of the current network ($Q(s, a, \theta)$) and target network ($Q_{tar}(s, a, \theta')$). The Q value of the target network is calculated differently from the DQN-PER-TEAE algorithm. Specifically, in the DDQN-PER-TEAE algorithm, the rule for updating the Q value of the target network is: if episode terminates at time step $t + 1$, then update according to Equation (19), otherwise update according to Equation (20). By decoupling the action selection and the calculation of the target Q value, the DDQN-PER-TEAE algorithm can reduce the overestimation bias in the DQN-PER-TEAE algorithm. Additionally, the incorporation of the TEAE model allows for the real-time adaptation of the exploration probability and selection of actions, further improving the efficiency and effectiveness of the algorithm.

$$Q_{tar}(s_t, a_t, \theta') = r_t. \tag{19}$$

$$Q_{tar}(s_t, a_t, \theta') = r_t + \gamma Q_{tar}(s_{t+1}, a', \theta'), \tag{20}$$

where $a' = \arg\max_{a'} Q(s_t, a'_t, \theta)$ is the action that can obtain the maximum Q value in the current network, and a'_t represents the action candidate set in state s_t.

Obviously, as with the DQN-PER-TEAE algorithm, we defined the TD error of the DDQN-PER-TEAE algorithm, as shown in Equation (21).

$$\delta(t) \doteq Q_{tar}(s_t, a_t, \theta') - Q(s_t, a_t, \theta), \tag{21}$$

where θ' and θ are the network parameters of the target and current network, respectively. And the formula for calculating the Q value of the current network is shown in Equation (22).

$$Q(s_t, a_t, \theta) = r_t + \gamma Q(s_{t+1}, a_{t+1}, \theta). \tag{22}$$

The pseudocode of the DDQN-PER-TEAE algorithm is similar to the DQN-PER-TEAE algorithm, except for changes to the network parameters and update formulas. In DDQN-PER-TEAE, the current and target network parameters are represented by θ and θ', and the update formulas in lines 10 and 16 are changed to Equations (21), (19) and (20), respectively. Compared to the DDQN-PER algorithm, DDQN-PER-TEAE avoids over-estimation and poor model robustness, and allows the agent to explore or exploit actions based on their environmental knowledge. As a result, the algorithm improves the efficiency and utility of empirical data.

4. Evaluation and Experiment

In this section, we will introduce the evaluation metrics used to assess the experiments, present the obtained experimental results, and provide a detailed analysis of the findings.

4.1. Evaluation Metrics

The experiment employs three primary evaluation metrics, namely, confidence score, standard score, and historical maximum score, to assess the performance of the proposed model and algorithms in solving the MDP. Analyzing these three metrics provides a comprehensive understanding of the ability and efficacy of the proposed approach.

Confidence Score: the confidence score is a quantitative evaluation of the probability of estimating the true distribution of the entire sampled data. It is used to assess the "trustworthiness" of the proposed model and algorithms, and to judge their stability, robustness, and generalization. In this experiment, we use a 95% confidence level to calculate the upper limit A and lower limit B of the confidence interval. A confidence coefficient of $Z = 1.96$ corresponds to a 95% confidence level. The upper limit A and lower limit B are calculated using the following formula, where $\overline{G}$ is the average of the agent scores, $\overline{\overline{G}}$ is the standard deviation of the agent scores, and N is the number of epochs set in this algorithm (which is 400 in this case).

$$A = \overline{G} + Z\frac{\overline{\overline{G}}}{\sqrt{N}}, \tag{23}$$

$$B = \overline{G} - Z\frac{\overline{\overline{G}}}{\sqrt{N}}. \tag{24}$$

Standard Score: The process of standardizing data, including specific game scores, involves applying a linear transformation to the original data so that it is mapped between the range of 0 and 1. The purpose of standardization is to reduce the magnitude of the data and make it easier to observe and analyze the changing trends more clearly. By standardizing the data, we can compare and assess the convergence of the proposed model and algorithms more effectively. The process of standardizing the specific game score is as follows:

$$G_s = \frac{G_a - G_r}{G_m - G_r}, \tag{25}$$

$$G_a = \sum_{t=0}^{n} \gamma r_t, \qquad t = 0, 1, 2, \cdots, n. \tag{26}$$

$$G_m = \max\{G_{a0}, G_{a1}, G_{a2}, \cdots, G_{aN}\}. \tag{27}$$

where G_a is the game score achieved by the agent trained with the algorithm selected for the experiment, which is the sum of the rewards r_t obtained by the agent at each time step after being discounted by the discount factor γ^t. G_r is a random score obtained by the agent, with specific values provided in the article by Google DeepMind [8]. G_m is the historical maximum score achieved by the agent in previous epochs, where G_{a1} represents the score achieved by the agent in the first epoch.

Historical Maximum Score: the historical maximum score is a statistical measure that represents the highest performance achieved by the agent throughout the training process. It provides valuable information about the potential of the proposed model and algorithms to excel in future experiments or real-world applications. The principle of the law of large numbers suggests that once a model or algorithm reaches its historical maximum score, it is likely to continue performing well in subsequent experiments [40]. By analyzing the historical maximum score, we can gain insights into the strengths and weaknesses of the proposed model and algorithms in terms of their overall performance.

4.2. Experimental Setup

The experiments were conducted on a server equipped with an Intel Xeon Platinum 8369HB CPU@3.30GHz Processor, 32 GB RAM, and Windows Server 2019 Operating System. We evaluated the proposed model and algorithms using the PyTorch deep learning architecture on the Atari 2600 RL benchmark. The Atari 2600 RL benchmark is part of the Arcade Learning Environment (ALE) [41,42], which is a widely used platform for testing and evaluating reinforcement learning methods. The ALE includes hundreds of Atari 2600 game environments, each of which is a MDP problem with unique challenges and

complexities. These game environments vary in terms of the dimensionality of their state and action spaces, providing a rigorous testing ground for comparing the performance of different RL algorithms. For more information about the Atari 2600 games, please refer to https://www.gymlibrary.dev/environments/atari/, accessed on 22 October 2022.

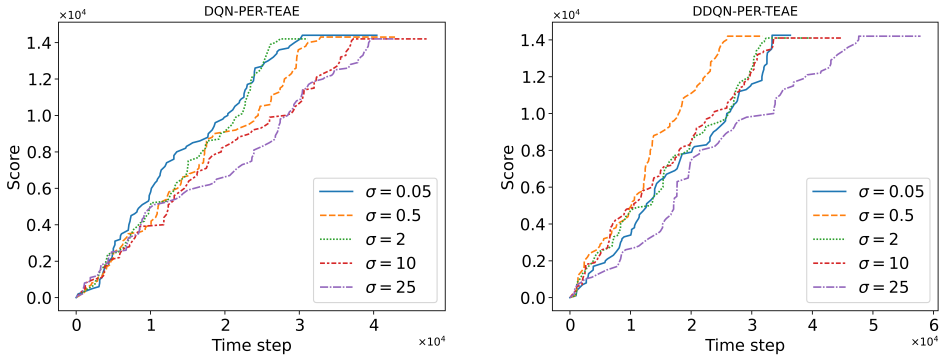
4.3. Experimental Results and Analysis

In this section, we initialize the main parts of the DQN-PER and DDQN-PER algorithms with the parameters set by DQN from Google DeepMind [8]. The TEAE model's initial exploration probability is set to 1, and the Adam optimizer is used with a learning rate of 1×10^{-5} during the training process [39]. We also use specific hyperparameter settings, as shown in Table 2.

Table 2. List of hyperparameters and their values.

Hyperparameter	Value	Description
Minibatch size	32	Number of training cases over which each stochastic gradient descent (SGD) update is computed.
Replay memory size	1×10^4	SGD updates are sampled from this number of most recent empirical data.
Update frequency	2	The frequency at which the target network parameters are updated.
Discount factor	0.99	Discount factor gamma used in the Q-learning.
Learning rate	1×10^{-5}	The learning rate used by Adam.
Initial exploration probability	1	Initial value of ϵ in TEAE model.
Number of epochs	4×10^2	4×10^2 epochs per algorithm iteration.
Time steps	5×10^3	5×10^3 time steps per epoch.

To investigate the impact of the inverse sensitivity coefficient σ on the performance of the TEAE model, we conducted experiments on the Kungfu Master game using the DQN-PER-TEAE and DDQN-PER-TEAE algorithms. The goal was to observe the effect of different σ values on the algorithm's convergence and assess the robustness of the TEAE model proposed in this paper. Figure 3 presents the experimental results.



(**a**) (**b**)

Figure 3. The influence of σ on the algorithm. Subfigure (**a**) and subfigure (**b**) respectively demonstrate the scores of DQN-PER-TEAE and DDQN-PER-TEAE under different σ values in the Kungfu Master game environment.

As shown in Figure 3a, the DQN-PER-TEAE algorithm achieves significantly faster convergence when $\sigma = 0.05$ in comparison to other σ values. Therefore, for the upcoming performance comparison experiments, we have selected an inverse sensitivity factor of 0.05 for the DQN-PER-TEAE algorithm. While $\sigma = 2$ may outperform $\sigma = 0.05$ in terms of scores achieved after roughly 25,000 time steps, $\sigma = 0.05$ leads to a more stable and faster optimization trend until convergence to relatively high scores is reached. Similarly, in Figure 3b, the DDQN-PER-TEAE algorithm achieves significantly better results at $\sigma = 0.5$

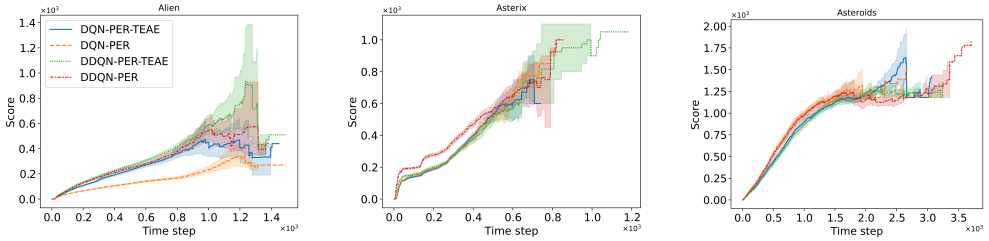
when compared to other σ values. Therefore, we have chosen an inverse sensitivity factor of 0.5 for the DDQN-PER-TEAE algorithm in the upcoming performance comparison experiments. Although $\sigma = 2$ and $\sigma = 10$ could initially lead to better scores than $\sigma = 0.5$, after about 15,000 time steps, $\sigma = 0.5$ allowed the algorithm to steadily improve until convergence to higher game scores.

Remark 1. *When keeping the parameters of the main algorithm constant, the values of σ (specifically, 0.05, 0.5, 2, 10, 25) have minimal impact on the TEAE model's performance in achieving high scores in Atari 2600 games. Moreover, based on the analysis conducted, it is evident that regardless of the chosen value of σ in the TEAE model, the algorithm consistently achieves commendable scores in the Kungfu Master game after a certain period of time. Therefore, it can be inferred that the TEAE model exhibits weak reliance on parameter values and demonstrates robustness in its performance.*

To further validate the effectiveness of the TEAE model, we conducted comparative experiments on six Atari 2600 games, namely Alien, Asterix, Asteroids, Battle Zone, Chopper Command, and Gopher. The goal was to compare the performance of two RL algorithms, namely DQN-PER-TEAE and DDQN-PER-TEAE, which incorporate the proposed adaptive exploration mechanism (TEAE) with state-of-the-art algorithms. On the model-free side, we selected DQN-PER [8,30], DDQN-PER [9,30], and SAC [10] as the benchmark algorithms. Additionally, we included the performance of uniform randomized exploration (RAND) for comparison. For each game and algorithm, a training process of 2 million iterations was initiated, followed by testing. Subsequently, confidence scores, standard scores, and historical maximum scores were calculated and analyzed.

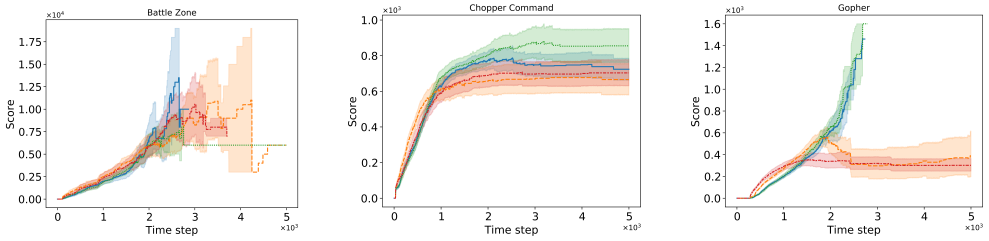
Figure 4 presents the training scores recorded for the four algorithms across 400 epochs. Each curve represents the scores at each time step within aepoch, with the solid line indicating the average score over the 400 epochs and the shaded area representing the 95% confidence interval obtained from 400 independent operational epochs. The results demonstrate the exceptional performance of the TEAE-based algorithms in the selected Atari 2600 games. These algorithms outperformed the annealing-based ϵ-greedy algorithm in four games and achieved comparable performance in the remaining two games, Asteroids and Battle Zone (Figure 4c,d). Moreover, both the DQN-PER-TEAE and DDQN-PER-TEAE algorithms exhibited significantly improved convergence speed. In the Alien, Asterix, Asteroids, and Battle Zone games (Figure 4a–d), the score curves displayed notable fluctuations. However, in the Chopper Command and Gopher games (Figure 4e,f), the score curves were relatively stable, with the Chopper Command game converging to an almost flat line. In the Gopher game, the score curve continued to rise. Despite these observations, the DQN-PER-TEAE and DDQN-PER-TEAE algorithms exhibited smoother score curves and narrower confidence intervals compared to the DQN-PER and DDQN-PER algorithms. This suggests that TEAE exhibits remarkable robustness and has the ability to enhance the robustness of benchmark algorithms.

Remark 2. *Under consistent basic parameter settings for the four RL algorithms, the benchmark algorithms based on the TEAE model exhibit small and smoothly fluctuating confidence intervals across the six Atari 2600 games. This observation indicates that the TEAE possesses excellent robustness and can enhance the robustness of the benchmark algorithms.*



(**a**) (**b**) (**c**)

Figure 4. *Cont.*



(**d**) (**e**) (**f**)

Figure 4. Confidence scores curve. The scores obtained by the four algorithms during the agent's training are recorded over a total of 400 epochs. The figure displays the scores for each time step within each epoch. The solid line represents the average score across the 400 epochs, while the shaded area represents the 95% confidence interval based on the scores obtained from the 400 independently epochs. The figure includes the scores for the following games: (**a**) Alien, (**b**) Asterix, (**c**) Asteroids, (**d**) Battle Zone, (**e**) Chopper Command, and (**f**) Gopher.

Tables 3 and 4 present a comparison between the proposed TEAE-based RL approaches and several non-TEAE-based RL methods. The results clearly demonstrate that TEAE-based RL outperforms other non-TEAE-based RL approaches in the Atari 2600 RL benchmark. In particular, we evaluate the performance based on the standard score. For the Alien, Asterix, and Chopper Command games, the best performing benchmark method among the non-TEAE-based RL approaches is SAC. However, the proposed DDQN-PER-TEAE method consistently outperforms SAC. For example, DDQN-PER-TEAE achieves a 1.2% improvement over SAC in the Alien game and a 0.6% improvement in the Chopper Command game. When considering the historical maximum score, SAC performs the best among the non-TEAE-based RL benchmark methods. Nevertheless, the proposed DDQN-PER-TEAE method consistently achieves superior performance compared to SAC. Specifically, DDQN-PER-TEAE surpasses SAC by 50 points in the Alien game, 90 points in the Chopper Command game, and 100 points in the Asterix game. Furthermore, we can observe that in the Alien game, DDQN-PER-TEAE achieves a score 610 points higher than DDQN-PEF, while DQN-PER-TEAE achieves a score 1180 points higher than DQN-PEF. In the Asterix game, DDQN-PER-TEAE outperforms SAC by 100 points. These data demonstrate the effectiveness of TEAE and its ability to enhance the performance of the benchmark algorithms to some extent.

Table 3. Comparison of TEAE and non-TEAE approaches regarding standard score.

	Alien	**Asterix**	**Asteroids**	**Battle Zone**	**Chopper Commmand**	**Gopher**
DQN-PER-TEAE	0.919	0.818	0.958	0.839	0.895	0.761
DDQN-PER-TEAE	0.976	0.955	0.971	0.649	0.897	0.833
DQN-PER	0.681	0.773	0.767	0.921	0.737	0.635
DDQN-PER	0.943	0.909	0.956	0.721	0.789	0.875
SAC	0.964	0.948	0.983	0.932	0.891	0.862
RAND	0.483	0.462	0.496	0.398	0.461	0.397

Table 4. Comparison of TEAE and non-TEAE approaches regarding historical maximum score.

	Alien	**Asterix**	**Asteroids**	**Battle Zone**	**Chopper Commmand**	**Gopher**
DQN-PER-TEAE	1930	900	2400	20,000	1700	1460
DDQN-PER-TEAE	2050	1100	2330	16,000	1900	1600
DQN-PER	750	850	1840	20,510	1400	1220
DDQN-PER	1440	1000	2300	18,000	1500	1920
SAC	2000	1000	2310	21,000	1810	2110
RAND	227.8	210	719.1	2360	811	257.6

Figure 5 displays the standard scores curves obtained by calculating the scores based on Equation (25) and the experimental results after 400 training episodes on six Atari 2600 games for the four algorithms. The overall test results indicate that the algorithms based on the TEAE model outperformed the other algorithms, with particularly notable improvements observed in the Alien, Asteroids, and Chopper Command games (Figure 5a,c,e). Additionally, in the Asterix game (Figure 5b), we can observe that the DDQN-PER-TEAE algorithm achieves higher scores compared to the DDQN-PER algorithm, and the DQN-PER-TEAE algorithm also outperforms the DQN-PER algorithm. These observations demonstrate that TEAE can effectively enhance the performance of benchmark algorithms, leading to improved game scores. Furthermore, the performance levels depicted in the figure illustrate the strong performance of the TEAE model and its capacity to enhance the overall performance of benchmark algorithms.

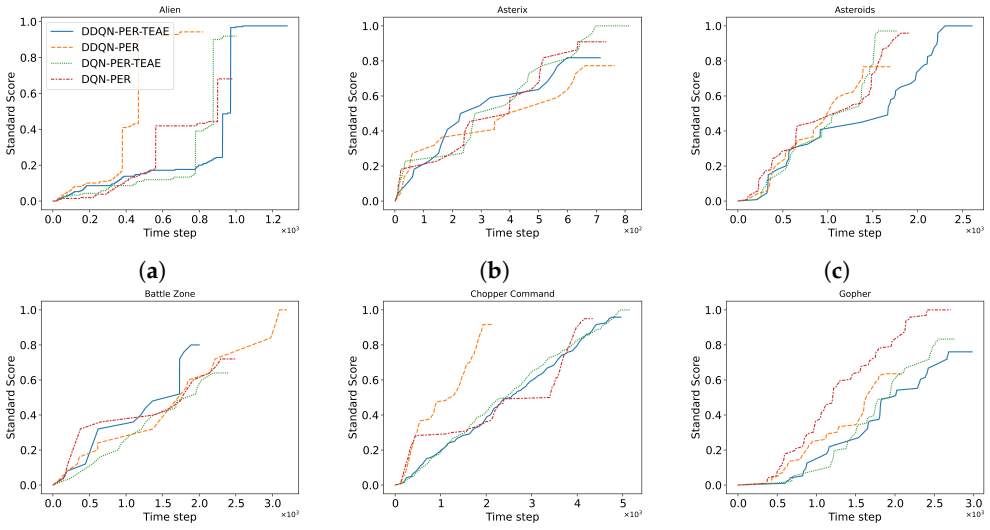


Figure 5. Standard scores curve. After 400 epochs of training the agent, the four algorithms were evaluated on a single epoch of testing. The game scores obtained were standardized using Equation (25). The figure includes the scores for the following games: (**a**) Alien, (**b**) Asterix, (**c**) Asteroids, (**d**) Battle Zone, (**e**) Chopper Command, and (**f**) Gopher.

Figure 6 provides a visual representation of the performance comparison between DDQN-PER-TEAE and DQN-PER-TEAE in the Atari 2600 games. It can be observed that DDQN-PER-TEAE consistently outperforms the benchmark algorithms in these games, and it exhibits the smallest performance gap compared to the human level. This observation suggests that the TEAE significantly improves the performance of the benchmark algorithm, further highlighting its ability to enhance the overall performance. Moreover, it is evident from Figure 6 that for each game, there is a significant difference between the performance of each algorithm and the human level. This discrepancy can be attributed to the challenge of accurately estimating the Q values in highly stochastic environments. When the Q values are estimated inaccurately, the TD error value becomes unreliable, leading to biased updates in the exploration probability. As a result, the performance of the TEAE may be indirectly affected. While the current work does not provide a solution to this problem, it is an area of consideration for future research and improvement.

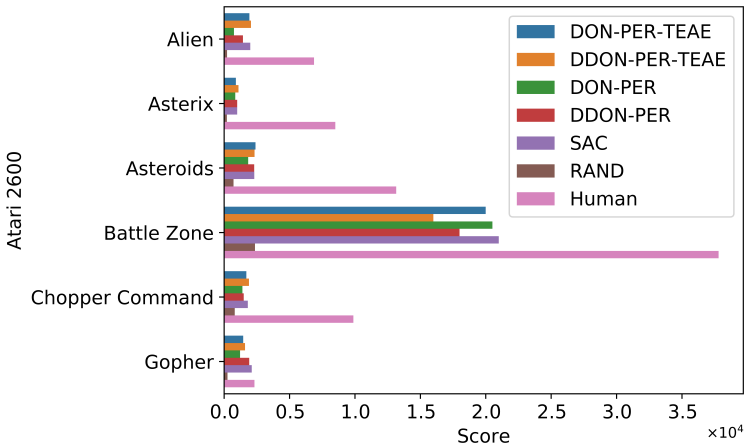


Figure 6. Historical maximum scores comparison chart. The chart compares the historical maximum scores achieved by the four algorithms during the training process for six Atari 2600 games.

Remark 3. *The benchmarking algorithm based on the TEAE consistently achieved higher scores in the Atari 2600 games, demonstrating the strong overall performance of the TEAE. These results indicate that the TEAE can effectively enhance the performance of the benchmarking algorithm, leading to improved game-playing capabilities and higher scores.*

4.4. Case Study

This section discusses the well-known Flexible Job Shop Scheduling Problem (FJSP) and applies the methods proposed in this paper to address these issues. In addition, we also conduct comparative experiments between the methods proposed in this paper and a solution approach based on Priority Dispatching Rules (PDR). The experiments demonstrate that the proposed method outperforms the traditional PDR, and exhibits high computational efficiency.

(1) FJSP [43]

An FJSP instance with dimensions $n \times m$ comprises $|J|$ jobs and $|M|$ machines, where $J = 1, 2, 3, ..., n$ and $M = 1, 2, 3, ..., m$. Each job $j_i \in J$ needs to undergo a series of operations, such as $o_{i1}, o_{i2}, \ldots, o_{in}$, in a fixed processing sequence to be completed. The operations are denoted by o_{ij}, and each o_{ij} can be processed on any machine m_k from the available machine set $M_{ij}(M_{ij} \subseteq M)$. The processing time is represented as $T_{ij,k}$. Each machine can handle only one operation at a time. To solve the FJSP problem, it is necessary to assign each operation o_{ij} to an idle machine and determine its start time s_{ij}, with the objective of minimizing the maximum production time $P_{\max} = \max_{i,j} P_{ij}$, where P_{ij} is the completion time of o_{ij}.

(2) Methodology

In this section, solving the FJSP is regarded as a sequential decision-making process. It iteratively takes scheduling actions, assigning an operation to an available machine, until all operations are processed. The workflow of the proposed method is depicted in Figure 7. In each iteration, tasks are initially initialized and transformed into a graph structure. Subsequently, the graph is inputted into a multi-layer deep convolutional neural network (DQN-PER-TEAE or DDQN-PER-TEAE) to extract features of operations and machines. These features are then utilized to generate a probability distribution of actions, ultimately yielding the optimal scheduling operation.

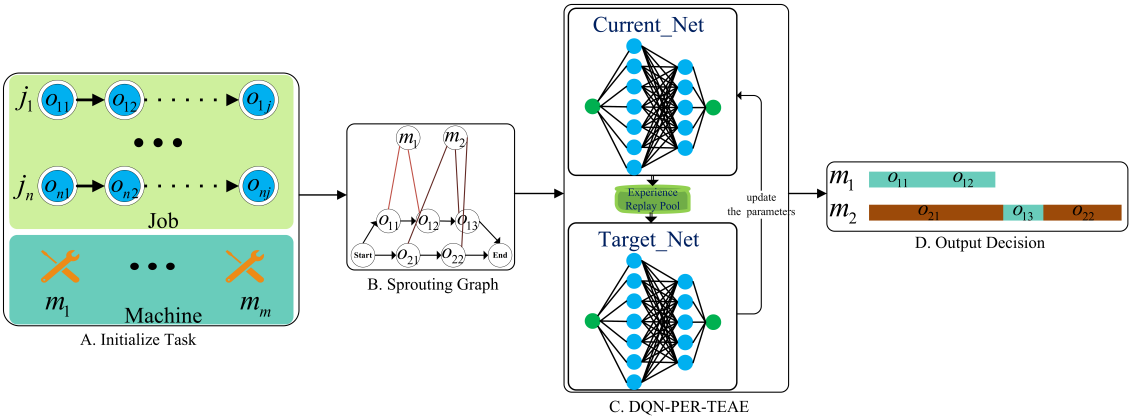


Figure 7. The workflow diagram for solving the FJSP.

(3) Markov Decision Process

State: The state s_t at time step t comprises all the operations and machines involved. The initial state s_0 is randomly selected from a distribution of FJSP instances. It is important to note that for each s_t, a operating time sequence $S(t)$ is maintained, calculated as follows: if o_{ij} is scheduled, its start time $S_{ij}(t)$ is the actual value of s_{ij}. If the preceding operation o_{il} of the same job is scheduled on machine m_k, then $S_{ij}(t) = s_{il} + T_{il,k}$; otherwise, $S_{ij}(t) = s_{il} + \hat{T}_{il}$, where $\hat{T}_{il} = \sum_{m_k \in M_{ij}} \frac{T_{il,k}}{|M_{ij}|}$ is the estimated processing time of o_{il}.

Action: In this case, a composite action method is employed to solve the FJSP, combining the selection of job operations and machine allocations into a single composite action. Specifically, an action a_t at time step t is defined as a feasible composite action (o_{ij}, m_k) at step t, where o_{ij} is executable and $m_k \in M_{ij}$ is available.

State Transition Function: Starting from state s_t and executing action a_t, the environment transitions to a new state s_{t+1}. For each task, a new state is entered upon the completion of each operation, until all operations are completed.

Reward: The reward is defined as the difference in the longest operating time, i.e., $r(s_t, a_t) = P_{\max}(S(t)) - P_{max}(S(t-1))$. When the discount factor $\gamma = 1$, the cumulative reward during one solving process is $R = \sum_{t=0}^{\tau} r(s_t, a_t) = P_{\max}(S(0)) - P_{\max}$, where τ is the time when the last operation of all jobs is completed.

Policy: The policy $\pi(a_t|s_t)$ defines a probability distribution over the action set a_t for each state s_t. In this case study, the algorithm proposed in this paper will be used to optimize the policy π towards maximizing the expected cumulative reward.

(4) Results

We initiated our evaluation by testing the proposed DQN-PER-TEAE and DDQN-PER-TEAE methods on three problems sourced from the XWdata dataset [44]. In parallel, we compared these approaches with two baseline RL methods, namely DQN-PER and DDQN-PER, alongside three well-known Priority Dispatching Rules (PDR) methods: FIFO (First In First Out), SPT (Shortest Processing Time), and LPT (Longest Processing Time) [45], all conducted within the same experimental environment. For each problem, we sampled test instances from the same distribution as the training data and conducted ten solution runs

to derive the optimal solutions. Table 5 provides an overview of the average solving time for different solving methods. The proposed methods consistently outperformed all PDR methods across the three problems. Among the PDR methods, FIFO demonstrated the highest efficiency, with an average solving time of 154.38 s for the three problems. In contrast, the most efficient RL method was DDQN-PER-TEAE, with an average solving time of 134.30 s, resulting in a time savings of 20.08 s compared to the FIFO. Furthermore, it is noteworthy that the proposed methods exhibited superior solving efficiency across all three problems compared to the baseline RL methods. This underscores the enhanced solving efficiency conferred by the proposed TEAE model.

To further assess the solving efficiency of the proposed methods on large-scale instances, we conducted tests on two problems from the HUdata dataset [44]. The results of the average solving time for different solving methods are presented in Table 6. It is evident that the advantages of the proposed methods are consistently maintained even on these large-scale instances. Among the RL solving methods, DDQN-PER-TEAE remains the top-performing approach. Furthermore, the TEAE-based RL methods exhibit a clear superiority over the baseline RL methods. These data affirm that the methods proposed in this paper are effective and efficient, both for small-scale and large-scale instances.

Table 5. Results on the XWdata.

Problem ($n \times m$)	DQN-PER -TEAE	DDQN-PER -TEAE	DQN-PER	DDQN-PER	FIFO	SPT	LPT
8×8	111.52	105.16	120.13	110.27	119.63	129.16	130.12
10×10	147.16	137.28	154.29	145.18	158.32	161.24	173.54
15×10	173.19	160.47	183.36	178.26	185.18	198.25	208.72

Table 6. Results on the HUdata.

Problem ($n \times m$)	DQN-PER -TEAE	DDQN-PER -TEAE	DQN-PER	DDQN-PER	FIFO	SPT	LPT
20×10	258.73	247.36	279.26	260.59	257.16	261.51	274.69
30×10	321.19	312.47	335.25	321.47	338.53	347.56	368.43

5. Conclusions

In this study, we propose a novel approach that balances exploration and exploitation based on the agent's learning process. Our aim is to approximate the optimal expected payoff function for subsequent sub-processes within the MDP to solve complex MDP problems more accurately. Leveraging the advantages of TD error feedback and dual-network architecture, we develop the DQN-PER-TEAE and DDQN-PER-TEAE algorithms, which incorporate an adaptive $\epsilon(s_t)$ greedy exploration model called TEAE. We extensively evaluate the effectiveness and robustness of the proposed model and algorithms on multiple evaluation metrics and uncertain Markov game environments. The experimental results demonstrate that the TEAE model achieves adaptive exploration, leading to improved learning efficiency. The RL algorithms based on TEAE exhibit outstanding performance and robustness, surpassing benchmark methods and even surpassing existing MDP solving approaches.

This study provides a suite of approximate models and algorithms based on RL decision making for addressing MDP problems. While the proposed approach has shown effectiveness, there is still room for improvement and extension in future research. Firstly, we plan to combine the TEAE model with other sampling techniques or RL algorithms to further enhance algorithmic performance. Secondly, we are interested in extending or adapting the proposed framework to tackle other social management, complex engineering optimization problems and intelligent decision-making problems.

Author Contributions: Conceptualization, X.W. and Z.Y.; methodology, X.W. and Z.Y.; software, Z.Y.; validation, Z.Y.; formal analysis, X.W.; investigation, Z.Y.; resources, X.W.; data curation, Z.Y.; writing—original draft preparation, X.W. and Z.Y.; writing—review and editing, X.W., Z.Y., G.C. and Y.L.; visualization, X.W.; supervision, X.W.; project administration, X.W.; funding acquisition, X.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Wuhan University and Wuhan University of Science and Technology: the National Natural Science Foundation of China (72031009) and Major projects of National Social Science Foundation of China (20&ZD058).

Data Availability Statement: The data can be shared upon request.

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

References

1. Xie, S.; Zhang, Z.; Yu, H.; Luo, X. Recurrent prediction model for partially observable MDPs. *Inf. Sci.* **2023**, *620*, 125–141. [CrossRef]
2. White, D. Infinite horizon Markov decision processes with unknown or variable discount factors. *Eur. J. Oper. Res.* **1987**, *28*, 96–100. [CrossRef]
3. Liu, Z.; Khojandi, A.; Li, X.; Mohammed, A.; Davis, R.L.; Kamaleswaran, R. A Machine Learning–Enabled Partially Observable Markov Decision Process Framework for Early Sepsis Prediction. *INFORMS J. Comput.* **2022**, *1*, 176–182. [CrossRef]
4. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
5. Bellman, R.; Kalaba, R.E. *Dynamic Programming and Modern Control Theory*; Academic Press: New York, NY, USA, 1965.
6. Puterman, M.L. Markov decision processes. In *Stochastic Models*; Elsevier: Amsterdam, The Netherlands, 1990; pp. 331–434.
7. Bellman, R. Dynamic programming. *Science* **1966**, *153*, 34–37. [CrossRef]
8. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
9. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the Thirtieth the Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; pp. 2094–2100.
10. 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 Thirty-Fifth International Conference on Machine Learning, Stockholm, Sweden, 25–31 July 2018; pp. 1861–1870.
11. Wang, D.; Ren, J.; Ha, M. Discounted linear Q-learning control with novel tracking cost and its stability. *Inf. Sci.* **2023**, *626*, 339–353. [CrossRef]
12. Bertsekas, D. *Reinforcement Learning and Optimal Control*; Athena Scientific: Nashua, NH, USA, 2019.
13. Wang, X.; Yang, Z.; Liu, Y.; Chen, G. A reinforcement learning-based strategy updating model for the cooperative evolution. *Phys. A* **2023**, *618*, 128699. [CrossRef]
14. Gosavi, A. Reinforcement learning: A tutorial survey and recent advances. *INFORMS J. Comput.* **2009**, *21*, 178–192. [CrossRef]
15. Nguyen, T.T.; Nguyen, N.D.; Nahavandi, S. Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications. *IEEE Trans. Cybern.* **2020**, *50*, 3826–3839. [CrossRef]
16. Zai, W.; Wang, J.; Li, G. A Drone Scheduling Method for Emergency Power Material Transportation Based on Deep Reinforcement Learning Optimized PSO Algorithm. *Sustainability* **2023**, *15*, 13127. [CrossRef]
17. Leon, J.F.; Li, Y.; Martin, X.A.; Calvet, L.; Panadero, J.; Juan, A.A. A Hybrid Simulation and Reinforcement Learning Algorithm for Enhancing Efficiency in Warehouse Operations. *Algorithms* **2023**, *16*, 408. [CrossRef]
18. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **2020**, *588*, 604–609. [CrossRef]
19. Lv, Z.; Tong, X. A Reinforcement Learning List Recommendation Model Fused with Graph Neural Networks. *Electronics* **2023**, *12*, 3748. [CrossRef]
20. Wu, X.; Huang, S.; Huang, G. Deep Reinforcement Learning-Based 2.5D Multi-Objective Path Planning for Ground Vehicles: Considering Distance and Energy Consumption. *Electronics* **2023**, *12*, 3840. [CrossRef]
21. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
22. Fehér, Á.; Aradi, S.; Bécsi, T. Online trajectory planning with reinforcement learning for pedestrian avoidance. *Electronics* **2022**, *11*, 2346. [CrossRef]
23. Huang, F.; Deng, X.; He, Y.; Jiang, W. A novel policy based on action confidence limit to improve exploration efficiency in reinforcement learning. *Inf. Sci.* **2023**, *640*, 119011. [CrossRef]
24. Yao, Z.; Yu, J.; Zhang, J.; He, W. Graph and dynamics interpretation in robotic reinforcement learning task. *Inf. Sci.* **2022**, *611*, 317–334. [CrossRef]

25. Guo, Z.; Thakoor, S.; Pîslar, M.; Avila Pires, B.; Altché, F.; Tallec, C.; Saade, A.; Calandriello, D.; Grill, J.B.; Tang, Y.; et al. Byol-explore: Exploration by bootstrapped prediction. *Adv. Neural. Inf. Process. Syst.* **2022**, *35*, 31855–31870.
26. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the Thirty-Fifth International Conference on Machine Learning, Stockholm, Sweden, 25–31 July 2018; pp. 1587–1596.
27. Zaks, G.; Katz, G. ReCom: A deep reinforcement learning approach for semi-supervised tabular data labeling. *Inf. Sci.* **2022**, *589*, 321–340. [CrossRef]
28. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
29. 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]
30. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
31. Triche, A.; Maida, A.S.; Kumar, A. Exploration in neo-Hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks. *Neural Netw.* **2022**, *151*, 16–33. [CrossRef] [PubMed]
32. Kyoung, D.; Sung, Y. Transformer Decoder-Based Enhanced Exploration Method to Alleviate Initial Exploration Problems in Reinforcement Learning. *Sensors* **2023**, *23*, 7411. [CrossRef]
33. Yuan, Y.; Yu, Z.L.; Gu, Z.; Yeboah, Y.; Wei, W.; Deng, X.; Li, J.; Li, Y. A novel multi-step Q-learning method to improve data efficiency for deep reinforcement learning. *Knowl. Based. Syst.* **2019**, *175*, 107–117. [CrossRef]
34. White, C.C.; White, D.J. Markov decision processes. *Eur. J. Oper. Res.* **1989**, *39*, 1–16. [CrossRef]
35. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
36. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
37. Meuleau, N.; Bourgine, P. Exploration of multi-state environments: Local measures and back-propagation of uncertainty. *Mach. Learn.* **1999**, *35*, 117–154. [CrossRef]
38. Kakade, S.M. *On the Sample Complexity of Reinforcement Learning*; University of London: London, UK, 2003.
39. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
40. Durrett, R. *Probability: Theory and Examples*; Cambridge University Press: Cambridge, UK, 2019; Volume 49.
41. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]
42. Machado, M.C.; Bellemare, M.G.; Talvitie, E.; Veness, J.; Hausknecht, M.; Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.* **2018**, *61*, 523–562. [CrossRef]
43. Adams, J.; Balas, E.; Zawack, D. The shifting bottleneck procedure for job shop scheduling. *Manag. Sci.* **1988**, *34*, 391–401. [CrossRef]
44. Xia, W.; Wu, Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Comput. Ind. Eng.* **2005**, *48*, 409–425. [CrossRef]
45. Wang, L.; Hu, X.; Wang, Y.; Xu, S.; Ma, S.; Yang, K.; Liu, Z.; Wang, W. Dynamic job-shop scheduling in smart manufacturing using deep reinforcement learning. *Comput. Netw.* **2021**, *190*, 107969. [CrossRef]