International Journal of Innovative Computing, Information and Control
Volume **20**, Number **6**, December **2024**

ICIC International ©2024 ISSN 1349-4198
pp. **1603–1620**

A STUDY OF GENERAL TRANSFER LEARNING OPTIMIZATION ALGORITHM FOR REINFORCEMENT LEARNING APPLIED TO VIDEO GAMES

Yuanzhi Huo[1], Mengjie Jin[2] and Sicong You[3]

[1]College of Information Engineering
[2]School of Mathematics and Statistics
Henan University of Science and Technology
No. 263, Kaiyuan Avenue, Luolong District, Luoyang 471000, P. R. China
{ 9906588; 9906479 }@haust.edu.cn

[3]College of Food Science and Technology
Nanjing Agricultural University
No. 1, Weigang, Xuanwu District, Nanjing 210095, P. R. China
2020208020@stu.njau.edu.cn

Received February 2024; revised June 2024

Abstract. *Reinforcement learning (RL) has demonstrated remarkable success in mastering complex tasks, particularly within the realm of video games, like the intelligent non-player characters (NPCs) and adaptive difficulty levels enhance player experience, providing more realistic and engaging gameplay. RL excels in training agents to engage in gameplay, generally, it is necessary to train distinct agents for each new game, this process needs considerable time costs for training and some training is redundant, especially for games of similar types. To address this challenge, transfer learning (TL) emerges as a promising solution. By leveraging knowledge acquired from one gaming scenario, a pre-trained agent can be fine-tuned or directly applied to a new game, this approach significantly reducing the training time required. However, when dealing with multiple games, determining which game serves as the source environment and which games serve as the target environment to achieve great TL results is a notable drawback which needs to be solved. In order to solve the aforementioned issues, we propose an innovative TL optimization algorithm. This algorithm can concurrently execute multiple games and screen out which game is suitable as the source environment and target environment based on the reward of different epoches. Agents trained on the source environment can be directly or fine-tuned and applied to new games. Meanwhile, the proposed algorithm can proceed a comparative analysis of various RL algorithms consequently obtaining the optimal RL algorithm for TL. In the experiment, we leverage two video games and utilize six classical RL algorithms as input. The results demonstrate through our proposed approach, the TL performance in the target environment attains an impressive reward equivalent to 99.98% of the RL reward.*

Keywords: Reinforcement learning, Deep learning, Transfer learning, Markov decision process, Actor-critic methods

1. **Introduction.** Reinforcement learning (RL) stands as a paradigm within the broader field of machine learning that has garnered significant attention for its remarkable ability to enable intelligent agents to learn optimal decision-making strategies in dynamic and complex environments [1, 2]. Unlike traditional supervised learning, where explicit examples guide the model, RL relies on agents interacting with an environment, receiving feedback in the form of rewards or penalties, and autonomously learning optimal actions

DOI: 10.24507/ijicic.20.06.1603

through trial and error [3]. The background of RL is inspired by the way humans and animals learn from their experiences. RL agents explore their environment, take actions, receive feedback, and continuously refine their strategies to maximize cumulative rewards over time. This process of learning from interactions is particularly potent in realm where explicit rules are challenging to define, and the optimal strategy may evolve with changing environments. One of the characteristics of RL is its applicability to a wide range of realms, including robotics, finance, healthcare, and notably video games.

Video games provide a complex and controlled environment ideal for testing and improving RL algorithms. They offer a diverse range of challenges that necessitate sophisticated decision-making and strategic planning. By training and testing RL algorithms in game settings, researchers can observe algorithm performance, identify potential issues, and enhance overall efficacy. The real-time feedback inherent in game environments further facilitates the rapid adjustment and optimization of RL strategies, significantly advancing the state-of-the-art in algorithm performance. Video game environments also serve as fertile grounds for theoretical exploration and validation in RL. They present complex decision-making problems that are instrumental in studying multi-agent collaboration, autonomous learning, and other advanced concepts. Moreover, the reward mechanisms in games offer valuable insights into the design and optimization of reward functions, which are crucial for the development and fine-tuning of RL algorithms. Thus, games not only test existing theories but also drive the evolution of new theoretical frameworks in RL. The dynamic and challenging nature of games stimulates the exploration of new algorithms and techniques. This includes advancements in deep reinforcement learning, meta-learning, and other cutting-edge approaches. The interdisciplinary nature of RL research in gaming fosters innovation by integrating knowledge from computer science, neuroscience, cognitive science, and other fields. This cross-pollination of ideas accelerates the development of novel methodologies and fosters a collaborative research environment. The application of RL in gaming also yields significant social and economic benefits. In the gaming industry, intelligent non-player characters (NPCs) and adaptive difficulty levels enhance player experience, providing more realistic and engaging gameplay. These advancements not only meet the growing market demand for sophisticated gaming experiences but also drive economic growth within the industry. Furthermore, the technologies and methodologies developed through gaming research often transfer to other sectors, generating broader economic impacts.

The ability of RL agents to master complex tasks within video games has showcased the potential for this approach to achieve impressive performance in strategic decision-making and adaptability. While RL has achieved notable success, several issues still need to be explored, such as the substantial training time and the requirement to retrain agents for different tasks or environments. In response to these challenges, researchers have explored innovative solutions with transfer learning (TL) emerging as a particularly promising avenue [4].

Unlike traditional learning approaches that start each task from scratch, TL leverages knowledge acquired from one environment called source environment to enhance the learning process in a related but different environment [5]. This approach holds significant promise in accelerating the adaptation of models to new tasks, reducing the need for extensive data, and fostering more efficient learning across a range of applications. The fundamental idea behind TL is inspired by the observation that knowledge gained from mastering one environment can be beneficial in improving performance in a different but related environment. In essence, TL seeks to capitalize on the inherent relationships and commonalities between environments to facilitate a more seamless transition of knowledge and skills [6]. The applications of TL span various fields, from computer vision and

natural language processing to robotics and video games. By enabling models to build upon existing knowledge, TL addresses the challenge of insufficient data, particularly in environments where data collection is expensive, time-consuming, or impractical. This not only enhances the adaptability of models but also contributes to the development of more robust and generalized artificial intelligence systems [7].

TL in RL has demonstrated significant success across various games. One notable example is the application of deep reinforcement learning, specifically Deep Q Networks (DQN), to play Atari 2600 games [8]. In 2016, DeepMind's research team achieved impressive results by training agents on simpler games and transferring the learned knowledge to more complex ones. This approach significantly accelerated the learning process, allowing agents to attain high-level performance across a range of games. In real-time strategy games such as StarCraft II, TL techniques have been employed to leverage knowledge gained from one game to expedite the learning process in another. This adaptability is crucial in mastering the intricate strategies and skills required in such dynamic environments. Similarly, in open-world environments like Grand Theft Auto V [9], TL has been applied to train agents for tasks such as driving and navigating urban landscapes. The insights gained from one scenario enable the agent to quickly adapt and excel in diverse environments, showcasing the versatility of TL. Even in seemingly simple games like Flappy Bird [10], TL has proven effective. Agents trained on one instance of the game can generalize their flying skills to perform well in similar games, illustrating the broad applicability of TL across different gaming contexts. However, despite these successes, existing examples often do not specify the methods or optimized algorithms used to determine the source and target environments. They lack detailed explanations on how many games are used for training in the source environment and how many in the target environment. Normally, there are four types approaches to selecting source environment [11].

- **Implicit:** This approach is only used within the same domain, such as tasks with the same action and state spaces, but possibly different goals. Therefore, it is usually possible to reuse the source task without worrying about negative transfer.
- **Human-Specified:** The agent assumes that the source task is manually selected and provided as input to the transfer method; hence, a human user/designer is required to perform the source task selection.
- **Human-Assisted:** A human provides some information to help estimate the similarity between tasks. The source task selection is then performed automatically based on this information.
- **Autonomous:** The agent estimates the similarity between tasks without additional information and autonomously selects the most promising source tasks.

In the approaches described above, the selection of source environments and target environments is still primarily based on human subjective choice and intuition. For simple tasks, this approach is feasible, but for complex tasks such as large-scale games, it is better for the agent to automatically select the source and target environments. However, optimizing the selection of source and target environments requires an optimization algorithm to determine the best choices. In this paper, we propose a general TL optimization algorithm for RL applied to video games. This algorithm can concurrently execute multiple games and screen which game is suitable as the source environment and which as the target environment based on the reward across different epochs. The proposal optimization algorithm compares the cumulative reward values of different RL algorithms in different epochs to select the optimal source and target environments, as well as the optimal RL algorithm. This optimization not only streamlines the process but also conserves resources, making it a valuable contribution to the field.

The rest of this paper is organized as follows. Section 2 introduces related work in literature. Section 3 introduces a summary of necessary knowledge for RL and TL. Section 4 presents the details of the proposed TL optimization algorithm. Section 5 presents the experiment environment, scenario and results. Section 6 presents the conclusion and future work.

2. **Related Work.**

2.1. **RL for video game.** In recent years, many researchers explored RL applied to video games, and most of them obtained significant progress. In [12], Lample and Chaplot introduce an architecture for deep reinforcement learning in 3D first-person shooter (FPS) games, addressing decision-making in partially observable states, unlike previous works that mainly dealt with fully observable 2D environments. The authors propose an approach to augmenting models with game feature information, such as the presence of enemies or items, during the training phase, alongside minimizing a Q-learning objective, which significantly improves training speed and agent performance. However, the model relies on internal variables provided by the game engine during training, which is limited in practical applications. At the same time, the model performs well on the ViZDoom platform, its generalizability to other FPS games or more complex environments has not been verified. In [13], Vinyals et al. introduce SC2LE (StarCraft II Learning Environment), a reinforcement learning environment based on the game StarCraft II. StarCraft II is a real-time strategy (RTS) game that combines fast-paced micro-actions with the need for high-level planning and execution. The paper outlines the observation, action, and reward specifications for the StarCraft II domain and offers an open-source Python-based interface for communication with the game engine. In addition to the main game maps, a suite of mini-games focusing on different gameplay elements is provided. Initial baseline results for neural networks trained on this data to predict game outcomes and player actions are presented. The paper concludes with initial baseline results for canonical deep reinforcement learning agents applied to the StarCraft II domain. However, the high complexity of the environment makes it difficult for existing algorithms to be directly applied. Meanwhile, even the basic mini-games present a high learning challenge for current reinforcement learning algorithms. In [14], Torrado et al. describe the interfacing of the General Video Game AI (GVGAI) competition and its associated software framework with the OpenAI Gym environment, a widely used interface for connecting agents to reinforcement learning problems. The authors evaluate the performance of several widely used implementations of deep reinforcement learning algorithms on a number of GVGAI games and analyze the results to indicate the relative difficulty of these games in comparison to each other and to those in the Arcade Learning Environment (ALE) under similar conditions. However, there is a significant variation in the performance of learning algorithms across different games, with some games showing good performance and others not. At the same time, training and testing on the GVGAI framework require substantial computational resources. On the other hand, the GVGAI framework, written in Java and communicating with Python through a local port, results in slower training compared to ALE. In [15], Rajeswaran et al. introduce a new framework that casts model-based reinforcement learning (MBRL) as a game between 1) a policy player, which attempts to maximize rewards under the learned model and 2) a model player, which attempts to fit real-world data collected by the policy player. A near-optimal policy for the environment can be obtained by finding an approximate equilibrium for the aforementioned game, and two families of algorithms are developed to find the game equilibrium by drawing upon ideas from Stackelberg games. Experimental studies suggest that the proposed algorithms

achieve state-of-the-art sample efficiency, match the asymptotic performance of model-free policy gradients, and scale gracefully to high-dimensional tasks like dexterous hand manipulation. While the algorithmic framework is proposed, the actual implementation is complex and requires fine-tuning. Meanwhile, Although the algorithms perform well in experiments, their generalization ability across a broader range of environments or tasks needs further validation.

2.2. **Transfer learning approach.** In [16], Lazaric focuses on developing methods to transfer knowledge from a set of source tasks to a target task, thereby improving the performance of learning algorithms. The author provides a formalization of the general transfer problem, identifies the main settings, and reviews significant approaches to transfer in RL. However, the paper lacks practical application cases, which would not be conducive to readers applying the theory to practice. Although it presents a variety of transfer methods, it does not delve into the specific implementation details of each method. In [17], Zhao et al. provide a systematic review of the challenges and methods associated with transferring deep reinforcement learning (DRL) policies from simulated environments to real-world robotics applications. It underscores the significance of DRL in robotic control, the challenges of real-world data collection, and the role of simulation in robotic training while highlighting the negative impact of the discrepancy between simulation and reality on policy transfer performance. Although it provides an overview of various methods, it does not offer enough comparative analysis to show the relative advantages and disadvantages of each approach. Meanwhile, the article does not cover all possible transfer learning scenarios, such as applications in different types of robots or tasks of varying complexity. In [18], Varghese and Mahmoud provide a comprehensive overview of the advancements in multi-task learning within deep reinforcement learning, comparing key approaches such as DISTRAL, IMPALA, and PopArt, and discussing challenges like scalability, distraction dilemma, and catastrophic forgetting. While offering valuable perspectives on different methods, the paper lacks in-depth technical details and comparative analysis between the methods and does not cover all possible application scenarios. In [19], Ng et al. delve into the theory of policy invariance when rewards are transformed and its practical application in reward shaping. It presents experimental results in grid worlds of varying sizes to demonstrate how policy invariance can be maintained through reward shaping techniques under changes to the reward function, with results visually displayed in charts. While the paper offers deep theoretical analysis and valuable insights, it have limitations in the scope of experimental design and the breadth of practical application, not fully showcasing the applicability and effectiveness of policy invariance across a range of tasks or complex environments. In [20], Wiewiora et al. discuss the incorporation of expert knowledge into reinforcement learning in a structured manner, especially as we scale up to real-world tasks. The paper presents a method for integrating arbitrary advice into the reward structure of a reinforcement learning agent without altering the optimal policy. This approach extends the potential-based shaping method proposed by Ng et al. [19] to shaping functions based on both states and actions, allowing for more specific guidance for the agent in choosing actions, without requiring the agent to discover this from the rewards on states alone. While the paper offers in-depth theoretical analysis and valuable insights, it have limitations in experimental design and the breadth of practical application, not fully demonstrating the applicability and effectiveness of policy invariance across a variety of tasks or complex environments.

2.3. **Application of TL in RL.** In [21], Sinapov et al. present a framework for selecting source tasks to improve the performance of target tasks in reinforcement learning without having samples from the target task. It uses metadata associated with each task to learn

the expected benefit of transfer between tasks. Tested through large-scale experiments in the Ms. Pac-Man domain, where an agent played over 170 million games across 192 variations of the task, the method successfully selected appropriate source tasks for previously unseen target tasks. While the study has achieved certain results without target task samples, it mainly considered the transfer from a single source task and did not delve into the scenario of transferring from multiple source tasks. In [22], Isele et al. introduce a lifelong learning approach that integrates task descriptors with coupled dictionary learning for zero-shot knowledge transfer, enhancing the performance of learned task models. By using task descriptors as feature vectors, the method enables rapid initiation of the learning process for new tasks without the need to collect training data on the actual tasks. However, shortcomings of the paper include a strong reliance on the choice and quality of task descriptors, as inaccurate or incomplete descriptors could affect learning efficacy. Furthermore, while the experiments show the method's effectiveness on dynamic control problems, its generalizability to other types of tasks or more complex scenarios needs further validation. In [23], Braylan and Miikkulainen introduce a transfer learning approach in the domain of video games to address the challenge of training game agents with limited data. By decomposing games into objects, learning object models, and transferring models from known to unfamiliar games to guide learning, the method has shown improved prediction accuracy and more efficient exploration over a comparable control, thus accelerating the training of game agents. However, shortcomings of the paper include insufficient discussion on the strategy for selecting source objects in the transfer learning process and a lack of in-depth analysis on the generalizability and applicability of the transfer method across different types of games. In [24], Fitzgerald and Bullard explore a situated mapping approach for transfer learning, addressing the object mapping challenge when transferring task models from a source to a target environment with potential novel objects. It proposes an interactive mapping by demonstration method that leverages guidance from a human teacher to predict the remaining object mappings based on a limited number of correspondences provided by the teacher. The approach was evaluated in both simulation and two physical tasks, sorting and assembly, demonstrating that an agent can quickly infer correct object mappings with minimal human assistance. However, shortcomings of the paper include a high dependency on perception systems and knowledge bases in real-world tasks, which could limit the universality and scalability of the approach.

3. **Background.**

3.1. **Markov Decision Process.**

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making problems where an agent interacts with an environment over a sequence of discrete time steps. The key feature of an MDP is the Markov property, which states that the future state of the system depends only on the current state and action, and not on the sequence of events that preceded them. This property simplifies the modeling of complex systems by focusing on the most recent information [25]. The MDP is defined by the tuple (S, A, P, R) where

- S is the set of states.
- A is the set of actions.
- P is the state transition probability function: $P(s'|s, a)$ represents the probability of transitioning to state s' given the current state s and action a.
- R is the reward function: $R(s, a, s')$ represents the immediate reward obtained after taking action a in state s and transitioning to state s'.

3.2. **Reinforcement learning.** Reinforcement learning (RL) is a branch of machine learning that focuses on how an agent learns to make a sequence of decisions to maximize cumulative rewards through interaction with an environment. In RL, the agent learns by interacting with the environment, observing its states, taking actions, receiving rewards, and learning how to maximize cumulative rewards over time. RL can be seen as a framework that utilizes MDP to model decision-making problems. MDP provides the formalism for RL problems, defining the structure of the environment, possible actions, and the consequences of those actions. RL algorithms, such as Q-learning and policy gradient methods, operate within the MDP framework to learn the optimal policy. The core concepts in RL are as follows [26]:

- State (s): represents a specific configuration or situation in the environment.
- Action (a): represents the possible actions the agent can take in a given state. The action set is denoted by A, including all possible actions.
- Policy (π): represents the probability distribution of taking specific actions in a given state. Denoted by π, where $\pi(a|s)$ is the probability of taking action a in state s.
- Reward (r): represents the immediate feedback the agent receives from the environment after taking an action.
- Value function: there are two types of value function.

  The first type is the state value function ($V_\pi(s)$): represents the expected cumulative reward starting from state s under policy π. The equation is shown as follows.

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s \right] \tag{1}$$

  The second type is action value function ($Q_\pi(s,a)$): represents the expected cumulative reward starting from state s, taking action a under policy π. The equation is shown as follows.

$$Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a \right] \tag{2}$$

The goal of RL is to find an optimal policy π^* that maximizes the expected cumulative reward. This is achieved by maximizing the value functions:

$$V^*(s) = \max_\pi V_\pi(s) \tag{3}$$

$$Q^*(s,a) = \max_\pi Q_\pi(s,a) \tag{4}$$

where $V^*(s)$ and $Q^*(s,a)$ represent the optimal state value and action value under the optimal policy.

3.3. **Transfer learning.** The integration of TL into RL aimed at adapting knowledge from a source environment to utilize it in a target environment. One crucial aspect is the consideration of the source and target domains, each characterized by its own state and action spaces, reward, and dynamics. This transition necessitates addressing the distributional differences between the two environments, often expressed through mathematical representations like the Kullback-Leibler (KL) divergence, denoted as

$$D_{KL}(P_{\text{source}} || P_{\text{target}}) \tag{5}$$

where P_{source} and P_{target} are the probability distributions in the source and target environments, respectively.

TL in RL often involves the transfer of policies between environments. The mathematical formulation for policy transfer aims to minimize the disparity between the source and target policies, typically expressed as

$$D_{KL}(\pi_{\text{source}}||\pi_{\text{target}}) \tag{6}$$

where π_{source} and π_{target} represent the policies in the source and target environments, respectively. This KL divergence quantifies the difference in the probability distributions of actions under the two policies. Incorporating Equation (5) and Equation (6) can achieve the trained model of the source environment directly or fine-tuning applying to the target environment.

4. Overview of TL and TL Optimization Algorithm.

4.1. Workflow of TL.

The workflow of TL is shown in Figure 1, which consists of three essential parts: input layer, intermediate layer, and output layer. The input layer plays a pivotal role in data input, and it can accept n games synchronously. The intermediate layer is dedicated to transferring knowledge from the source environment to the target environment. The determination of optimal RL algorithm, source and target environments are determined by the TL optimization algorithm proposed in this paper. The output layer serves the purpose of assessing the TL performance by using the optimal RL algorithm for the given TL scenario. The details of the workflow of TL are shown as follows.

1) Prepare n games for the input layer ($n \geq 2$).
2) Input all games to the TL optimization algorithm in the intermediate layer. Following this algorithmic procedure, the optimal RL algorithm, source and target environments can be determined.
3) The trained agent from the source environment is applied in the target environment. The trained agent can be directly utilized or fine-tuned.

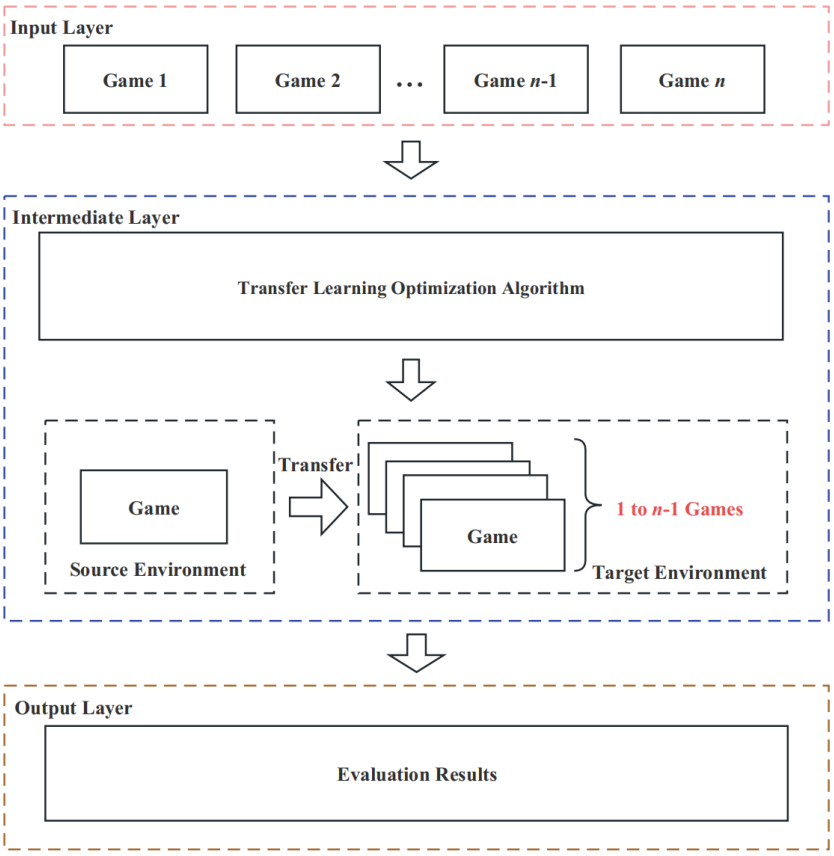


FIGURE 1. Workflow of TL

4) Evaluate TL performance using multiple different RL algorithms and confirm the proposed TL optimization algorithm is effective.

In the input layer, all input games must belong to the same or similar category, because diverse categories exhibit variations in operations, environments, and scoring mechanisms. Our research concentrates on TL within games of the same or similar category. Additionally, a minimum of two input games is required, as TL necessitates at least one source environment and one target environment.

Algorithm 1: TL Optimization Algorithm

Require: n_g games ($n_g \geq 2$), n_a RL algorithms, n_e epochs
Initialize $T_1 \leftarrow n_e/3$, $T_2 \leftarrow 2n_e/3$, $T_3 \leftarrow n_e$, $A \leftarrow n_e/2$
Randomly select one RL algorithm a_i from n_a
if a_i not in *tabu* **then**
  Put a_i in *tabu*
else
  Goto 2
end if
Put a_i in *tabu*
Run n_g games in parallel
for i in n_e **do**
  if i equals T_1 **then**
    Put $score_g i$ in RT_1
  end if
  if i equals T_2 **then**
    Put $score_g i$ in RT_2
  end if
  if i equals T_3 **then**
    Put $score_g i$ in RT_3
  end if
  if i equals A **then**
    Put $score_g i$ in RA
  end if
end for
if len(*tabu*) is not equal len(n_a) **then**
  Goto 2
end if
Normalize(RT_1, RT_2, RT_3)
$n_g i$ add 1 $\leftarrow$ argmaxRT_1, $n_g i$ add 1 $\leftarrow$ argmaxRT_2, $n_g i$ add 1 $\leftarrow$ argmaxRT_3
NormalizeRA
$a_{\max} \leftarrow$ argmaxRA in argmax$n_g i$
return $n_g i$ game, $a_{\max}$
Ensure: Source environment, Target environment, Optimal RL algorithm

4.2. **Procedure of TL optimization algorithm.** The procedure of the proposal TL optimization algorithm is shown in Algorithm 1, and the necessary symbols are defined as follows:

- n_g: represents input games.
- $n_g i$: represents single game in n_g, used for voting.

- n_a: RL algorithms which are utilized to train agent.
- n_e: training epoch of RL algorithms.
- $score_g i$: represents accumulative rewards obtained for $n_g i$.
- *tabu*: saves selected RL algorithm, and ensure that no duplication of RL algorithms in the *tabu* list.
- T_i: represents different epochs, each T_i represents $n_e/3$ $(i = 1, 2, 3)$.
- RT_i: saves $score_g i$ of T_i $(i = 1, 2, 3)$.
- A: represents $n_e/2$ epochs.
- RA: saves $score_g i$ of A.
- $a_{\max}$: optimal RL algorithm.

The procedure of the TL optimization algorithm can be delineated into three main parts. The first part spans from **line 1** to **line 9** of Algorithm 1. It is responsible for handling input data and initializing parameters. The input includes games, RL algorithms, and epochs, where epoch means the iteration number of training. In **line 1**, the epoch n_e is divided into one-third epoch, half epoch, two-thirds epoch and epoch itself. These values are assigned to T_1, A, T_2, and T_3, respectively. After the initialization of the parameters, the algorithm proceeds to randomly select an RL algorithm. Subsequently, it checks whether the selected algorithm is already used. If not, the algorithm puts it into the *tabu* set. Following these operations, n_g games begin to execute.

From **line 10** to **line 26**, the core optimization logic of the TL optimization algorithm is implemented. The algorithm enters a loop that iterates with n_e, within this loop, a "voting mechanism" is proposed in the main logic, where the entire epoch is subdivided into four phases: T_1, T_2, T_3, and A. These phases correspond to one-third of the epoch, two-thirds of the epoch, the entire epoch, and half of the epoch, respectively. As introduced in Section 1, intelligent agents in RL learn correct actions based on the reward values they receive. Different learning strategies can influence whether agents prioritize current or future rewards. For an optimal intelligent agent, we hope the goal is to maximize rewards both in the present and the future. To achieve this, the algorithm calculates cumulative reward values for each game during phases T_1, T_2, and T_3 and stores them in the list RT_1, RT_2 and RT_3. If the algorithm still has unused RL algorithms, proceed to line 2 and continue executing other algorithms.

From **line 27** to **line 31**, the algorithm screens out the maximum values of RT_1, RT_2, and RT_3 and increments the corresponding $n_g i$ for the game that achieves the maximum value by 1. When the training epoch reaches A, the algorithm saves the $score_g i$ RA. The logic for determining the termination of the algorithm is presented in this part. If all RL algorithms have been executed, the maximum value in RA can be output, indicating which RL algorithm has the highest score across the maximum $n_g i$ game. Finally, the algorithm returns which game should be the source environment, which games should be the target environment and which is the optimal RL algorithm.

Due to the different levels, stages, and reward mechanisms of different games, even games from different generations in the same series may have slight differences. In order to ensure a unified standard is used for comparison, we used min-max normalization in the algorithm to scale the score. The normalized formula is as follows.

$$X_{\text{normalized}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{7}$$

where:

- $X_{\text{normalized}}$: the normalized score.
- X: the original score.
- $X_{\min}$: the minimum value in score.

- $X_{\max}$: the maximum value in score.

Following the normalization process, all the score ranges within the interval of $[0, 1]$. Through this approach, all results can be compared using a unified standard.

5. Experiment.

5.1. Experiment environment and parameters.

In Table 1, we show the experiment environment and parameters. About experiment games, we choose Sonic the Hedgehog series [33]. Sonic the Hedgehog is a classic video game series developed by SEGA [34]. The initial version was first released in 1991. The series features Sonic, a super-fast blue hedgehog as the main character. The objective of the game is typically to collect golden rings, avoid enemies and obstacles, and ultimately defeat the enemies. In this paper, we choose the most classic two generations as the experiment game: Sonic the Hedgehog 1 and 2. These two generations due to their continuity have similar game styles, types, and operations, making them suitable as experimental subjects for TL. To achieve more comprehensive experimental results, we also choose six classic RL algorithms for training game agents and utilize Algorithm 1 to obtain which is optimal RL algorithm for TL. Each algorithm will be trained on 5 different training epochs and the results will be outputted optimal RL algorithms, source and target environment.

Table 1. Experiment environment and parameters

Parameter	**Value**	**Meaning**
Games	Sonic the Hedgehog	Input data
RL algorithms	A2C, PPO, DQN, DDQN, Rainbow, PG [27-32]	RL algorithms
Epoch	1×10^5, 2×10^5, 3×10^5, 4×10^5, 5×10^5	Training iterations

5.2. RL algorithm results.

From Table 2 to Table 5, we present the intermediate results of various RL algorithms applied to Sonic the Hedgehog 1 (Sonic 1) using the proposed TL optimization algorithm. In terms of horizontal analysis, both DQN and Double Deep Q-Network (DDQN) exhibit increasing scores as epochs progress. However, Advantage Actor-Critic (A2C) algorithm shows intermittent fluctuations, notably experiencing a significant decrease at 3×10^5 and 4×10^5 epochs of T_1. Proximal Policy Optimization (PPO) demonstrates a decrease only at 5×10^5 epoch of A, while Rainbow only decreases at 5×10^5 epoch of T_3. On the other hand, Policy Gradient (PG) shows fluctuation at 4×10^5 and 5×10^5 epochs.

Examining the results from the vertical analysis, PPO achieves optimal performance at 1×10^5 epoch of T_1 with a score of 2,458.63, while DDQN and PPO consistently outperform in the remaining epochs. At 2×10^5 epoch, PG attains optimal performance across T_1 to

Table 2. RL algorithm results of T_1 epoch for Sonic 1

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	1,383.89	4,076.92	4,177.28	3,959.68	**28,666.81**
PPO	**2,458.63**	2,578.79	2,708.07	7,705.94	8,103.85
DQN	1,269.72	3,685.68	6,015.58	9,878.13	11,290.45
DDQN	1,858.94	4,877.01	11,367.71	**22,480.48**	26,213.17
Rainbow	1,358.6	4,646.18	7,474.38	11,525.17	12,923.11
PG	453.68	**5,781.49**	**12,976.74**	6,702.92	6,409.7

TABLE 3. RL algorithm results of T_2 epoch for Sonic 1

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	3,760.53	13,677.03	8,037.66	8,661.37	55,768.74
PPO	7,008.45	12,888.43	16,827.81	29,749.84	32,935.34
DQN	5,081.92	14,456.61	21,134.35	34,867.58	39,864.53
DDQN	**8,209.46**	17,727.62	**32,687.72**	**52,945.63**	**62,917.06**
Rainbow	5,997.36	16,317.14	27,738.7	39,100.13	40,028.94
PG	5,197.33	**22,298.27**	28,410.27	13,464.63	12,460.56

TABLE 4. RL algorithm results of T_3 epoch for Sonic 1

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	5,476.08	22,093.89	11,898.05	13,378.91	86,428.19
PPO	13,336.25	19,654.84	25,733.81	39,374.32	46,253.04
DQN	11,979.52	29,301.55	42,511.34	58,924.42	66,214.35
DDQN	**14,715.15**	33,122.45	**55,865.27**	**85,232.75**	**101,222.52**
Rainbow	12,427.91	32,269.74	50,803.82	67,982.86	64,652.61
PG	11,768.01	**43,993.10**	51,071.78	19,153.23	18,464.46

TABLE 5. RL algorithm results of A epoch for Sonic 1

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	2,906.13	8,102.70	6,107.47	6,307.47	**44,537.54**
PPO	**5,626.85**	6,889.29	7,328.32	19,196.91	18,609.94
DQN	2,660.47	7,725.51	12,409.71	20,864.84	24,829.27
DDQN	4,285.99	10,632.38	**21,694.10**	**37,369.07**	44,234.41
Rainbow	3,349.27	9,383.75	16,536.36	23,652.28	24,946.18
PG	2,053.32	**11,151.88**	19,292.59	10,621.14	9,429.98

A. At 3×10^5 epoch, PG shows optimal performance at T_1, and DDQN shows optimal performance at the remaining epochs. At 4×10^5 epoch, DDQN shows superiority across all epochs. Finally, at 5×10^5 epoch, A2C stands out with optimal performance at T_1 and A, whereas DDQN dominates in the remaining epochs.

From Table 6 to Table 9, we present the intermediate results of various RL algorithms applied to Sonic the Hedgehog 2 (Sonic 2) using the proposed TL optimization algorithm. In terms of horizontal analysis, A2C and Rainbow demonstrate an increasing trend in scores as epochs progress. PPO exhibits notable performance across most epochs, with the exception of the 3×10^5 epoch of T_1. DQN consistently performs well across all epochs. DDQN shows fluctuations across nearly all at 4×10^5 epoch, excluding T_3. PG shows fluctuations across nearly all epochs at 3×10^5 epoch, excluding T_3.

Examining the results from the vertical analysis, DDQN achieves optimal performance at 1×10^5 epoch of T_1 with a score of 775.9, while PG consistently outperforms in the remaining epochs. PPO attains optimal performance across 2×10^5 epoch of T_1 to A. A2C attains optimal performance across 3×10^5, 4×10^5 and 5×10^5 epoch of T_1 to A.

5.3. **TL optimization and TL results.** In Section 4, we introduced the **vote mechanism** utilized by the proposed algorithm to determine the source and target environments

TABLE 6. RL algorithm results of T_1 epoch for Sonic 2

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	351.3	1,115.55	**24,075.67**	**46,326.72**	**52,606.84**
PPO	319.5	**13,272.83**	12,472.63	15,175.47	48,703.31
DQN	447.1	2,293.77	5,807.07	10,810.87	17,621.55
DDQN	**775.9**	2,396.48	4,215.62	3,333.81	9,556.1
Rainbow	435.38	2,395.94	5,029.1	11,899.06	20,782.57
PG	417.86	6,732.68	3,217.54	25,303.62	43,272.18

TABLE 7. RL algorithm results of T_2 epoch for Sonic 2

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	1,130.84	6,678.87	**63,214.55**	**97,728.87**	**117,703.96**
PPO	3,371.19	**41,572.68**	51,232.60	66,078.21	113,641.91
DQN	1,982.86	10,623.90	25,768.80	40,509.14	61,875.84
DDQN	2,898.48	10,661.30	14,682.39	10,669.16	32,041.16
Rainbow	2,527.22	12,590.54	23,759.37	42,100.68	65,861.74
PG	**8,739.04**	22,762.84	19,053.98	72,274.69	107,259.93

TABLE 8. RL algorithm results of T_3 epoch for Sonic 2

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	3,250.97	9,602.83	**102,365.73**	**149,162.48**	**182,875.46**
PPO	12,477.66	**70,033.25**	89,994.19	116,996.22	178,673.27
DQN	4,890.92	24,867.65	52,320.91	78,945.01	113,301.42
DDQN	3,843.86	19,930.43	23,039.98	30,169.20	70,983.92
Rainbow	6,941.76	28,940.52	48,200.03	80,776.83	118,304.04
PG	**23,100.14**	50,594.96	56,444.29	123,227.54	170,682.90

TABLE 9. RL algorithm results of A epoch for Sonic 2

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
A2C	665.51	3,495.95	**43,642.99**	**72,032.76**	**85,249.69**
PPO	1,275.52	**27,411.25**	31,780.10	40,624.11	81,164.40
DQN	1,258.96	5,454.41	14,624.78	23,998.62	38,109.70
DDQN	2,224.63	6,739.69	8,665.42	8,052.90	20,322.02
Rainbow	674.04	6,409.12	13,433.87	25,280.51	42,053.99
PG	**2,556.27**	13,458.21	7,780.84	46,827.98	75,642.74

and Table 10 presents the TL optimization algorithm results. Across ranging from 1×10^5 to 5×10^5 epoch, except for Sonic 1 and Sonic 2 exhibiting a **1 : 2** voting result at 1×10^5 and 3×10^5 epochs, the remaining epochs consistently show a **0 : 3** voting results. Consequently, Sonic 2 is determined as the source environment, while Sonic 1 is determined as the target environment. Based on the analysis in Table 10, it is evident that the optimal RL algorithms for various epochs are PG, PPO and A2C.

TABLE 10. TL optimization algorithm results

RL algorithms	1×10^5	2×10^5	3×10^5	4×10^5	5×10^5
T_1	Sonic 1	Sonic 2	Sonic 1	Sonic 2	Sonic 2
T_2	Sonic 2	Sonic 2	Sonic 2	Sonic 2	Sonic 2
T_3	Sonic 2	Sonic 2	Sonic 2	Sonic 2	Sonic 2
A	Policy Gradient	PPO	A2C	A2C	A2C

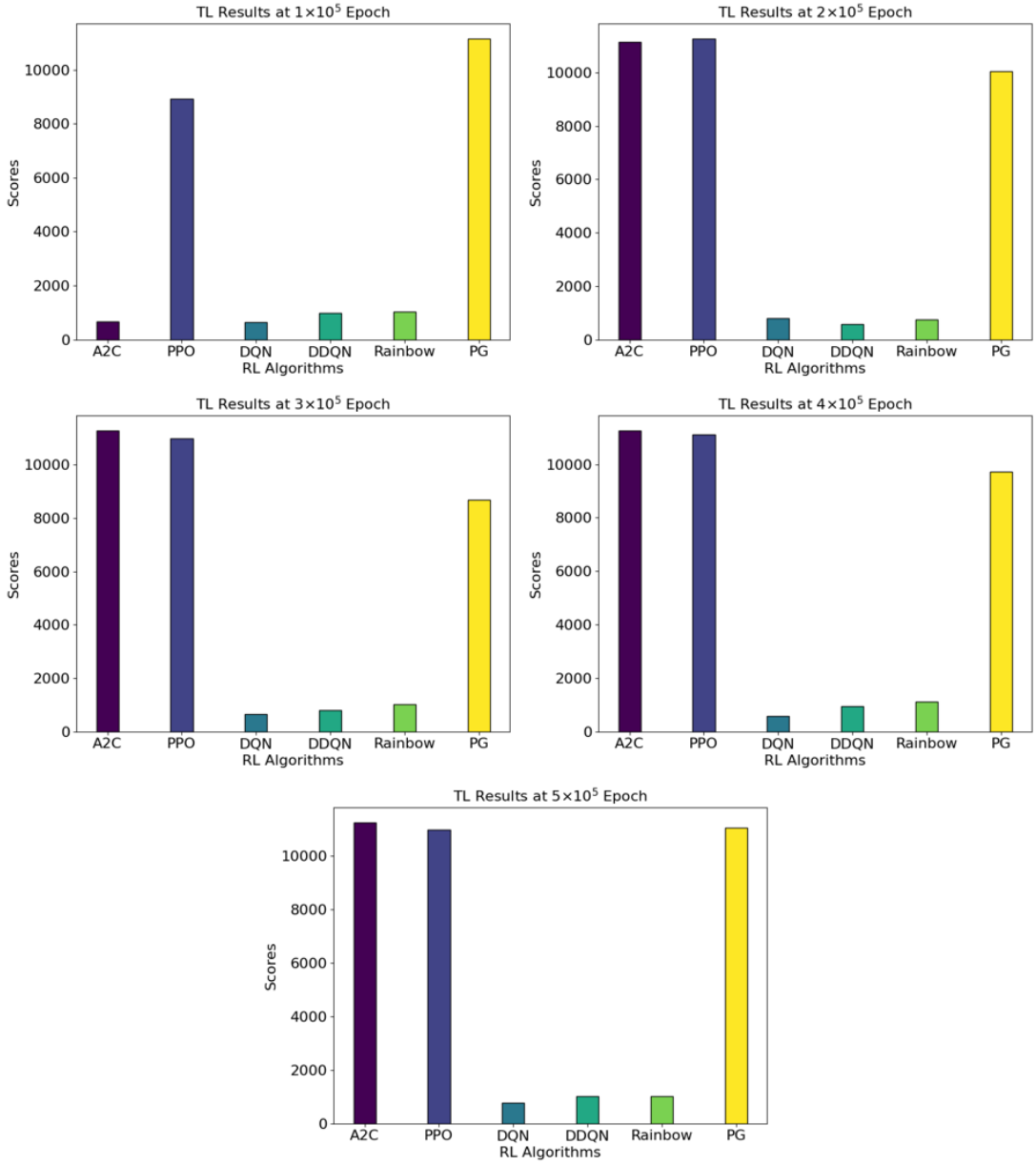


FIGURE 2. TL results of different RL algorithms at diverse epochs

In Figure 2, we present the TL results based on optimization results in Table 10. Sonic 2 is the source environment, while Sonic 1 serves as the target environment. All TL results correspond to the optimized results of various RL algorithms listed in Table 10. PG is the optimal RL algorithm at 1×10^5 epoch, PPO is the optimal RL algorithm at 2×10^5

epoch, and A2C is the optimal RL algorithm at 3×10^5, 4×10^5, and 5×10^5 epochs. The experiment iteration is 10,000 rounds.

At 1×10^5 epoch, PG obtains a score of 11,154.98, while PPO comes in the second with a score of 8,921.45. At 2×10^5 epoch, PPO obtains a score of 11,277.83, while A2C comes in the second with a score of 11,150.6. PG exhibits a fine difference from A2C, obtaining a score of 10,039.32. At 3×10^5 epoch, A2C obtains a score of 11,283.03, while PPO comes in the second with a score of 10,991.52. At 4×10^5 epoch, A2C obtains a score of 11,267.1, while PPO comes in the second with a score of 11,112.84. Finally, at 5×10^5 epoch, A2C obtains a score of 11,260.28, while PG comes in the second with a score of 11,057.5. PPO shows a fine difference compared to PG, obtaining a score of 10,984.56.

Based on the comprehensive analysis results, A2C, PPO, and PG demonstrated great performance in TL within the experimental gaming environment. Meanwhile, we know that the score may not consistently increase with the number of training epochs. In the context of the game environment discussed in the paper, impressive performance has already been demonstrated with A2C, PPO and PG. However, it is noteworthy that DQN, DDQN, and Rainbow did not exhibit remarkable performance in TL. The reasons are shown as follows.

1) A2C, PPO and PG are policy gradient-based methods that directly optimize the policy to learn optimal actions. This is in contrast to traditional value function optimization methods (DQN, DDQN, and Rainbow) that indirectly derive the policy by optimizing the value function. In TL scenarios, direct policy optimization can be more flexible.
2) Policy gradient methods are generally more robust to changes in the environment. When there are differences between the source and target environments, policy gradient methods may adapt more easily because they directly optimize the policy rather than relying on precise value function estimates.
3) In TL, the efficiency of utilizing samples is crucial for rapid learning in the new environment. Policy gradient methods often exhibit high sample efficiency because they can compute gradients with a single sample without the need for iteration or backup.
4) In certain situations, policy gradient methods may have a faster learning speed.

5.4. **Contrast experiment.** Using the optimal algorithm PG with 1×10^5 epoch as a case study, we conducted a comparative analysis between the average scores obtained from 10 TL times on Sonic 1 and the scores from the original RL results. The contrast experiment iterations are 100, 1,000 and 10,000 rounds, respectively.

In Figure 3, we present the contrast experiment results at 1×10^5 for different rounds. The results indicate a clear trend of increasing TL scores with the number of rounds, particularly noticeable at 10,000 rounds. In the 100-round scenario, there is a slight deviation between TL and original RL results, with the TL score reaching 84.85% of the original RL score. Progressing to 1,000 rounds, the difference between TL and original RL results diminishes, and the TL score reaches an impressive 99.91% of the original RL score. Finally, at 10,000 rounds, the TL score nearly matches the original RL score, achieving 99.98% of the original RL score.

Based on the aforementioned results, it becomes evident that as the agent is trained in extensive epochs, the TL results progressively converge towards the performance of the original RL results in the same environment. Ultimately, the TL scores approach parity with the original RL results.

6. **Conclusions.** In this paper, we explored the challenges associated with the conventional approach of training agents for video game in the realm of RL. While RL has shown remarkable success in mastering complex tasks within video games, however, the training

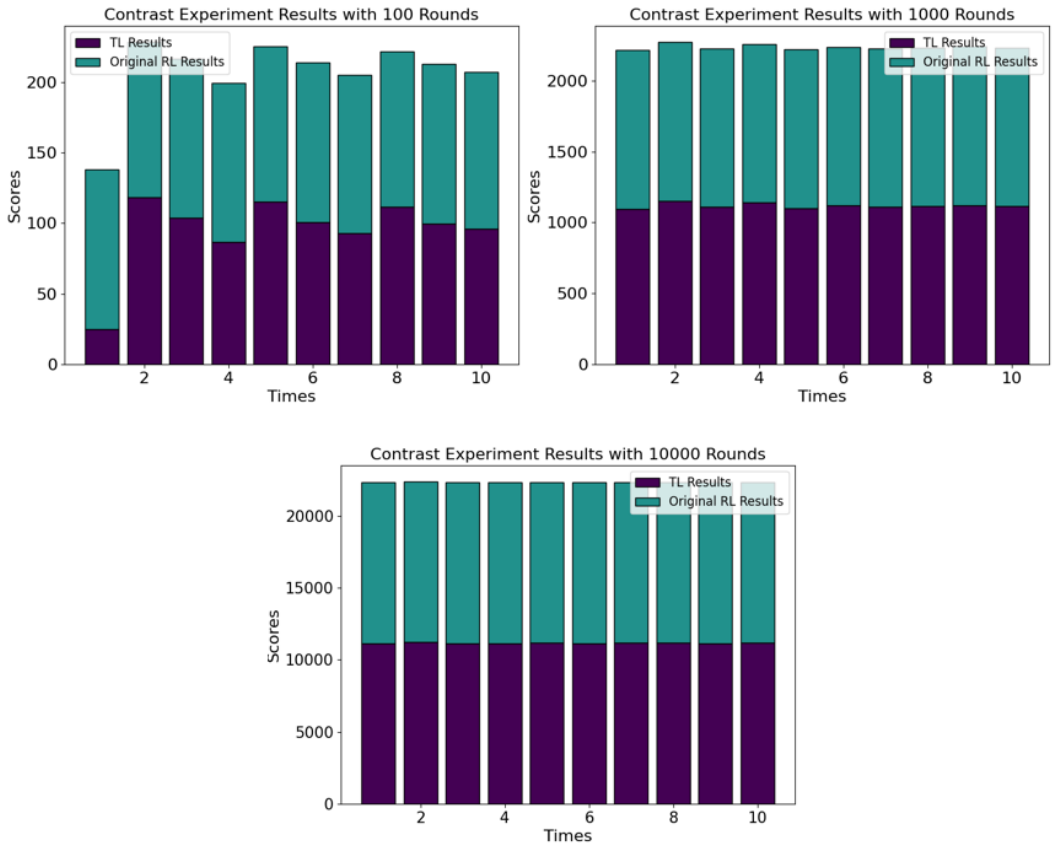


FIGURE 3. Contrast experiment results

process for each new game remains a notable drawback. To solve this issue, TL as a solution can train agents in the source environment and transfer knowledge and reuse policy to new games is a significant step forward. Nevertheless, how to determine the source and target environment is also an essential issue that directly influences TL performance.

Our proposed TL optimization algorithm addresses the critical issue of determining the source and target environments when dealing with multiple games. By executing multiple games and screening suitable environments based on RL algorithm performance and accumulative rewards, the proposed algorithm effectively outputs the optimal RL algorithm and source, target environment for TL. The agents trained on the source environment can be directly applied to new games. The experiment results demonstrated the practicality and efficiency of our approach. We utilized two video games and six classical RL algorithms as input and experiment results demonstrated that the TL performance in the target environment achieves an impressive 99.98% accumulation of rewards compared to RL results in the same environment, demonstrating the effectiveness of our approach. However, our proposed method still has the following issues. 1) We only evaluated one type of video game, and the number of games is not particularly large. 2) The proposed approach has not been evaluated on larger and complex games. 3) The proposed approach is mainly based on determining whether a game can serve as a source or target environment based on the reward values obtained at different epochs, due to the different training results of different RL algorithms, multiple comparative evaluations are required. Due to the need for multiple evaluations, the CPU time required will be longer.

In future work, we will attempt to explore the integration of the proposed approach with deep learning approaches, by extracting different features during game training, such as state space features, action space features, reward structures, and environmental

interaction frequencies, to select the source environment and target environment, thereby addressing the shortcomings of the proposed approach.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, MIT Press, Cambridge, 1998.

[2] G. Tesauro, TD-Gammon, a self-teaching backgammon program, achieves master-level play, *Neural Computation*, vol.6, no.2, pp.215-219, 1994.

[3] G. A. Vouros, Explainable deep reinforcement learning: State of the art and challenges, *ACM Computing Surveys*, vol.55, no.5, pp.1-39, 2022.

[4] I. Szita, *Reinforcement Learning in Games*, Springer, Berlin, 2012.

[5] M. E. Taylor and P. Stone, Transfer learning for reinforcement learning domains: A survey, *Journal of Machine Learning Research*, vol.10, no.56, pp.1633-1685, 2009.

[6] J. Ramon, K. Driessens and T. Croonenborghs, *Transfer Learning in Reinforcement Learning Problems through Partial Policy Recycling*, Springer, Berlin, 2007.

[7] F. Zhuang et al., A comprehensive survey on transfer learning, *Proceedings of the IEEE*, vol.109, no.1, pp.43-76, 2021.

[8] V. Mnih et al., Playing Atari with deep reinforcement learning, *arXiv Preprint*, arXiv: 1312.5602, 2013.

[9] M. A. Martinez et al., Beyond Grand Theft Auto V for training, testing and enhancing deep learning in self driving cars, *arXiv Preprint*, arXiv: 1712.01397, 2017.

[10] K. Yang, Using DQN and double DQN to play Flappy Bird, *Proc. of the 2022 International Conf. on Artificial Intelligence, Internet and Digital Economy (ICAID2022)*, pp.1166-1174, 2022.

[11] F. L. Da Silva and A. H. R. Costa, A survey on transfer learning for multiagent reinforcement learning systems, *The Journal of Artificial Intelligence Research*, vol.64, pp.645-703, 2019.

[12] G. Lample and D. S. Chaplot, Playing FPS games with deep reinforcement learning, *Proc. of the AAAI Conference on Artificial Intelligence*, vol.31, no.1, 2017.

[13] O. Vinyals et al., StarCraft II: A new challenge for reinforcement learning, *arXiv Preprint*, arXiv: 1708.04782, 2017.

[14] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu and D. Perez-Liebana, Deep reinforcement learning for general video game AI, *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, Maastricht, Netherlands, pp.1-8, 2018.

[15] A. Rajeswaran, I. Mordatch and V. Kumar, A game theoretic framework for model based reinforcement learning, *International Conference on Machine Learning*, 2020.

[16] A. Lazaric, Transfer in reinforcement learning: A framework and a survey, in *Reinforcement Learning. Adaptation, Learning, and Optimization*, M. Wiering and M. van Otterlo (eds.), Berlin, Heidelberg, Springer, 2012.

[17] W. Zhao, J. P. Queralta and T. Westerlund, Sim-to-real transfer in deep reinforcement learning for robotics: A survey, *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, Canberra, ACT, Australia, pp.737-744, 2020.

[18] N. V. Varghese and Q. H. Mahmoud, A survey of multi-task deep reinforcement learning, *Electronics*, vol.9, no.9, 1363, 2020.

[19] A. Y. Ng, D. Harada and S. Russell, Policy invariance under reward transformations: Theory and application to reward shaping, *Proc. of the 16th International Conf. on Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[20] E. Wiewiora, G. W. Cottrell and C. Elkan, Principled methods for advising reinforcement learning agents, *Proc. of the 20th International Conf. on Machine Learning*, pp.792-799, 2003.

[21] J. Sinapov et al., Learning inter-task transferability in the absence of target task samples, *Proc. of the 2015 International Conf. on Autonomous Agents and Multiagent Systems*, 2015.

[22] D. Isele, M. Rostami and E. Eaton, Using task features for zero-shot knowledge transfer in lifelong learning, *Proc. of the 25th International Joint Conf. on Artificial Intelligence*, New York, NY, pp.1620-1626, 2016.

[23] A. Braylan and R. Miikkulainen, Object-Model transfer in the general video game domain, *AIIDE*, vol.12, no.1, pp.136-142, 2021.

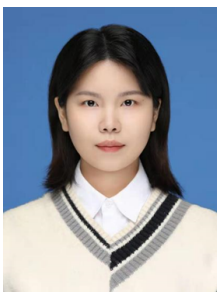
[24] T. Fitzgerald and K. Bullard, Situated mapping for transfer learning, *Proc. of the 4th Annual Conf. on Advances in Cognitive Systems*, Evanston, IL, pp.1-14, 2016.

[25] R. Bellman, A Markovian decision process, *Indiana University Mathematics Journal*, vol.6, no.4, pp.679-684, 1957.

[26] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction, *IEEE Transactions on Neural Networks*, vol.9, no.5, 1054, 1998.
[27] V. Mnih et al., Asynchronous methods for deep reinforcement learning, *ICML*, pp.1928-1937, 2016.
[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, Proximal policy optimization algorithms, *arXiv Preprint*, arXiv: 1707.06347, 2017.
[29] L. Lv, S. Zhang, D. Ding and Y. Wang, Path planning via an improved DQN-based learning policy, *IEEE Access*, vol.7, pp.67319-67330, 2019.
[30] H. Van Hasselt, A. Guez and D. Silver, Deep reinforcement learning with double Q-learning, *Proceedings of the AAAI Conf. on Artificial Intelligence*, vol.30, no.1, 2016.
[31] M. Hessel et al., Rainbow: Combining improvements in deep reinforcement learning, *Proceedings of the AAAI Conf. on Artificial Intelligence*, vol.32, no.1, 2018.
[32] R. S. Sutton, D. McAllester, S. Singh and Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, *Neural Information Processing Systems*, vol.12, pp.1057-1063, 1999.
[33] I. H. Chiang, C. M. Huang, N. H. Cheng, H. Y. Liu and S. C. Tsai, Efficient exploration in side-scrolling video games with trajectory replay, *The Computer Games Journal*, vol.9, no.3, pp.263-280, 2019.
[34] L. A. de Almeida and M. R. Thielo, An intelligent agent playing generic action games based on deep reinforcement learning with memory restrictions, *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Recife, Brazil, pp.29-37, 2020.

Author Biography

Yuanzhi Huo received B.S. degree in Computer Science from Zhengzhou University in 2017, the M.S. degree in Computer Science from Inner Mongolia University of Technology in 2020, the Ph.D. degree in the Division of Industrial Innovation Sciences from Okayama University, Japan in 2023. Since Dec. 2023, he has been a lecturer in the College of Information Engineering of Henan University of Science and Technology. He published 2 EI Compendex articles, 3 international conference articles, and 2 inventions. His research interests include reinforcement learning, deep learning, AI gaming, optimization algorithm, and computational fluid mechanics.



Mengjie Jin received B.S. degree in Mathematics and Applied Mathematics from Henan University of Science and Technology in 2017, the M.S. and Ph.D. degree from Hunan University in 2023. Since Jul. 2023, she has been a lecturer in School of Mathematics and Statistics of Henan University of Science and Technology. She published 2 SCI and 1 EI Compendex articles. Her research interests include non-Hausdorff topology and domain theory.

Sicong You received his B.S. and M.S. degrees in Food Technology from Henan University of Science and Technology in 2017 and 2020. Since 2020 to 2024, he is a Ph.D. course student in Food Science and Technology at Nanjing Agricultural University. In Jun. 2024, he received a Ph.D. degree. Since Sep. 2024, He has been a lecturer in College of Veterinary Medicine of Nanjing Agricultural University. He published 6 SCI articles. His research interests in non-destructive testing of agricultural products.