Kochi University of Technology

Enhancing Data Utilization in Game AI: Deep Reinforcement Learning Strategies for the Game 2048

by

Weikai Wang

Supervisor
Professor Kiminori Matsuzaki

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Engineering Course, Department of Engineering,
Graduate School of Engineering

September 2025

Abstract

Deep Reinforcement Learning (Deep RL) has achieved successful performance in various domains by combining deep neural networks with reinforcement learning algorithms, enabling agents to learn optimal policies directly from high-dimensional sensory inputs. Despite these advancements, efficient data utilization remains a critical challenge, particularly in complex environments with high-dimensional state spaces and stochastic dynamics.

This thesis focuses on enhancing Deep RL agents within the context of the game 2048—a single-player sliding block puzzle game that is easy to play but hard to master. 2048 presents significant strategic depth due to the high randomness of tile placements and the exponential growth of tile values. This level of randomness closely resembles the unpredictability found in real-world scenarios, making 2048 a valuable subject for study. Traditional neural network architectures, such as convolutional neural networks (CNNs), often struggle to capture the global contextual information which is important for strategic decision-making in 2048. Moreover, the high variance in the data caused by the randomness of the game can lead the training process to fall into local optima, further limiting performance.

To address these challenges, we investigate two primary research questions: (1) How does the implementation of global embeddings affect the performance of Deep RL agents in game AI? (2) How can refining evaluation function affect the performance of Deep RL agents in game AI? (3) How can reducing value gaps affect the performance of Deep RL agents in game AI?

We propose three main methods centered around enhancing the 2048 agent. First, we introduce global embedding techniques to improve the model's ability to process and utilize global information from the game state. This includes developing neural network architectures such as Bypass CNN (BP-CNN), Fully Connected Embedding CNN (FC-CNN), Convolutional Embedding CNN (CNN-CNN), and Transformer Embedding CNN (T-CNN), each designed to effectively integrate global embeddings.

Second, we develop refining evaluation functions aimed at achieving further performance improvement after the baseline model reaches a plateau. By reducing variance and incorporating deeper search strategies through methods such as Full-width TD-afterstate (TD-afterstate-full) and TD-afterstate with 2-ply search (2-ply TD-afterstate), these methods generate and utilize more accurate training data. This refinement enables the agent to go beyond the limitations of the standard TD-afterstate method, leading to higher scores.

Third, we focus on reducing value gaps, which arise from inconsistencies between 1-ply and 2-ply evaluations in the TD-afterstate training method. To address this issue, we developed the TDA-all method that updates all possible afterstates instead of only the best one. Although this approach improves exploration, it suffers from severe overestimation and fails to converge. To overcome this limitation, we introduced Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all), which separates action selection and evaluation by employing two networks. This design significantly alleviates overestimation, resulting in more stable training and controlled value estimates. Furthermore, when combined with TDA-full refinement, DDQN TDA-all achieves competitive performance with the best n-tuple based agents, demonstrating the effectiveness of reducing value gaps for enhancing DNN-based 2048 players.

Comprehensive experiments are conducted to assess the effectiveness of the proposed methods compared to baseline models. The results demonstrate that our methods significantly improve the scoring performance of Deep RL agents in 2048. These findings contribute to the advancement of Deep RL methodologies, with potential applications extending to other domains.

Acknowledgements

Time flies, and before I knew it, four years of my Ph.D. journey have passed.

First and foremost, I would like to express my deepest gratitude to my supervisor, Professor Kiminori Matsuzaki. The two published papers and the completion of my doctoral degree would not have been possible without his invaluable guidance. More importantly, however, is the research mindset I have learned from him. Looking back, I realize that when I first entered the university, I was not yet a qualified researcher, largely because I lacked the ability to explore and investigate the fundamental question of "why." As my research skills gradually improved, my way of thinking and understanding of many aspects of both research and life also changed significantly. These changes are inseparable from Professor Matsuzaki's guidance, which has been truly meaningful to me.

I would also like to thank other faculty members and staff at Kochi University of Technology. Their diverse perspectives inspired many of my thoughts, and their support helped me greatly in adapting to life in a foreign country.

My heartfelt thanks also go to my friends and family. The pressure and challenges faced during my Ph.D. studies could only be overcome with their constant support and companionship.

I am equally grateful to Kochi University of Technology and Kochi Prefecture. I deeply appreciate the environment here and the warmth of the people, both of which have been important motivations that sustained me throughout my doctoral journey.

Finally, I would like to express my gratitude to my home country for providing me with the opportunity and scholarship to study abroad. Conducting research in a different environment and culture has been an extremely valuable and effective experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Background

In recent years, the field of artificial intelligence (AI) has experienced significant advancements, particularly in deep learning and reinforcement learning. Deep Reinforcement Learning (Deep RL) combines the representation power of deep neural networks with the decision-making capabilities of reinforcement learning, enabling agents to learn optimal policies directly from high-dimensional sensory inputs. This synergy has led to remarkable achievements in various domains, including robotics, natural language processing, and notably, game AI.

Game AI serves as an ideal testbed for developing and evaluating AI algorithms due to its well-defined rules and measurable outcomes. The successes of DeepMind's AlphaGo and AlphaZero [2, 3] have demonstrated the potential of Deep RL agents to achieve superhuman performance in complex strategic games like Go, Chess, and Shogi. These achievements are largely attributed to the agents' ability to effectively utilize vast amounts of data to learn sophisticated strategies.

The game 2048 is a popular and challenging testbed for AI research. The game is easy to play but hard to master, making it still posing challenges for algorithmic agents. Unlike board games such as Go or Chess, 2048 is characterized by its randomness in tile generation and the necessity of long-term strategic planning. These properties closely resemble real-world decision-making scenarios, where outcomes are influenced by both stochastic factors and the ability to optimize over extended horizons. As such, 2048 provides a valuable platform for evaluating the robustness and generalization capabilities of reinforcement learning methods.

1.2 Research Question

The game 2048 is a single-player sliding block puzzle game that involves combining tiles with the same number to create higher-numbered tiles on a 4×4 grid. The game's simplicity in rules contrasts with its strategic depth, making it a compelling subject for AI research. Developing an AI agent capable of achieving high scores in 2048 requires sophisticated decision-making and efficient data utilization methods to handle the randomness of tile placements and the exponential growth of tile values.

Currently, methods based on N-tuple networks [4] have shown superior performance in the game 2048 compared to existing Deep RL approaches. This performance gap suggests that there are areas where Deep RL methods can be enhanced to better exploit the available data and improve learning efficiency.

Key areas for improvement in Deep RL approaches include:

- **Enhanced Utilization of Global Information:** Traditional neural network architectures, such as convolutional neural networks (CNNs), often focus on local feature extraction due to limited receptive fields. This limitation can hinder the network's ability to capture and utilize global contextual information essential for strategic decision-making in 2048. Enhancing the network's capacity to process global information may lead to better performance.
- **Reduction of Variance and Inaccuracy in Training Data:** The stochastic nature of the game introduces significant variance in the training data, which can lead to unstable learning and suboptimal policy estimation. Developing methods to generate and utilize more accurate data can improve the agent's learning process and performance.
- **Enhanced Exploration Ability:** Exploration is a crucial part in reinforcement learning. By strengthening the exploration capability of agents, models can avoid being limited to only the best immediate actions and instead better evaluate a wider range of possible afterstates. This can lead to more stable and effective learning.

These areas for improvement are not exclusive to Deep RL methods; traditional approaches like N-tuple networks may also face similar challenges. However, focusing on these aspects within Deep RL offers the potential to enhance its performance and possibly surpass existing methods in the game 2048.

This leads us to the central research question:

How can improved data utilization methods enhance the scoring performance of Deep Reinforcement Learning agents in game AI?

To address this overarching question, we focus on two sub-questions:

1. **Global Embeddings:** *How does the implementation of global embeddings affect the performance of Deep Reinforcement Learning agents in game AI?*
2. **Refining Evaluation Functions:** *How can agents achieve further performance improvement after reaching a performance plateau in game AI?*
3. **Enhanced Exploration Ability:** *How can improving the exploration ability of agents help to achieve more stable learning in game AI?*

1.3 Research Objectives and Significance

The primary objective of this research is to enhance the data utilization methods in Deep RL agents to improve their performance in the game 2048. Specifically, the research aims to:

1. **Enhance Global Information Utilization through Global Embeddings:** Develop neural network architectures that effectively integrate global embeddings to improve the model's ability to utilize global information present in the game state. This addresses the first sub-question by exploring the impact of global embeddings on agent performance.
2. **Improve Training Stability through Refining Evaluation Functions:** Design methods that generate and leverage more accurate evaluation data, such as Full-width TD-afterstate and 2-ply TD-afterstate, to enhance the stability and quality of training after the baseline performance reaches a plateau. This addresses the second sub-question by exploring how refined evaluation functions can achieve further performance improvement.
3. **Enhance Exploration Ability by Reducing Value Gaps:** Develop approaches such as TDA-all and DDQN TDA-all to update not only the best afterstate but also other possible afterstates, thereby mitigating overestimation and reducing

value gaps. This addresses the third sub-question by investigating how improved exploration ability contributes to more stable learning in game AI.

The significance of this research lies in its potential contributions to both theoretical and practical aspects:

- **Advancement of Deep RL Methodologies:** By focusing on data utilization, the research contributes to improving learning efficiency and effectiveness in Deep RL agents, providing insights into how agents can better leverage available data.
- **Enhancement of Game AI Performance:** Improved data utilization methods can lead to higher-scoring agents in 2048 and potentially other games, demonstrating the practical benefits of the research in developing more competent AI agents.
- **Broader Applications:** The findings can inform data utilization strategies in other domains where efficient and accurate data usage is critical, such as robotics, finance, and autonomous systems.

1.4 Overview of Research Methods

To address the research questions, we propose three main methods centered around enhancing data utilization:

1.4.1 Global Embedding Techniques

We introduce global embedding techniques to improve the model's utilization of global information in the game state. By embedding global contextual information into the neural network architecture, we aim to provide the agent with a more comprehensive understanding of the game environment, leading to better decision-making.

The implementation involves integrating global embeddings into the baseline CNN22 model, resulting in enhanced architectures such as:

- **Bypass CNN (BP-CNN):** Incorporates positional information directly into the input layer by concatenating positional embeddings, enabling the network to utilize spatial context effectively.

- **Fully Connected Embedding CNN (FC-CNN):** Utilizes fully connected layers to create a global embedding vector that encapsulates the overall state of the game board, which is then integrated with the convolutional layers.
- **Convolutional Embedding CNN (CNN-CNN):** Adds convolutional layers dedicated to processing and embedding global information, leveraging convolutional operations to capture complex patterns across the entire board.
- **Transformer Embedding CNN (T-CNN):** Employs a Transformer Encoder to capture global dependencies through self-attention mechanisms, effectively modeling the relationships between all tiles on the board.

These models are designed to investigate how the implementation of global embeddings affects the agent's performance by enhancing the utilization of global data, directly addressing the first sub-question.

1.4.2 Refining Evaluation Functions

We develop advanced training methodologies focused on refining evaluation functions to train the agent. By reducing variance and incorporating deeper search strategies, we aim to improve the stability and quality of the training data, enhancing the agent's learning process.

The methods include:

- **Full-width TD-afterstate (TD-afterstate-full):** Reduces variance in the training data by considering all possible future states from a given afterstate, leading to more accurate value estimates and better utilization of available data.
- **TD-afterstate with 2-ply search (2-ply TD-afterstate):** Extends the search depth to two plies, allowing the agent to account for a broader range of future scenarios and utilize deeper information in the data.

1.4.3 Reducing Value Gaps

We design methods aimed at reducing value gaps, which arise from difference between 1-ply and 2-ply evaluations in the TD-afterstate framework. Exploration is a crucial component in reinforcement learning, and insufficient exploration often results in agents

focusing only on the best afterstate, leaving other afterstates underutilized. This imbalance leads to value gaps and unstable learning. By enhancing the exploration ability of agents, we aim to improve the stability of training.

The methods include:

- **All-updated TD-afterstate (TDA-all):** Extends the TD-afterstate update rule by updating the values of all possible afterstates instead of only the best one. This encourages broader exploration and reduces the reliance on a single afterstate, but may suffer from severe overestimation in practice.
- **Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all):** Addresses the overestimation problem of TDA-all by separating action selection and action evaluation through two neural networks. This method alleviates value inflation, resulting in more stable learning and controlled value estimates.

1.5 Contributions

The main contributions of this thesis can be summarized as follows:

- **Global Embedding:** We propose some enhanced neural network architectures (BP-CNN, FC-CNN, CNN-CNN, and T-CNN) that use global embeddings into the baseline CNN22 model, showing their effectiveness in improving the agent's ability to utilize global information and outperforming our baseline method.
- **Refining Evaluation Functions:** We introduce evaluation refinement techniques, including TD-afterstate-full and 2-ply TD-afterstate, that reduce variance in training data and improve the stability of learning. These methods enable agents to achieve further performance improvements even after the baseline has plateaued.
- **Reducing Value Gaps:** We identify and analyze the value gap problem in TD-afterstate training and propose solutions, including TDA-all and DDQN TDA-all. The latter effectively mitigates overestimation issues and, when combined with TDA-full refinement, results in more stable learning and controlled value estimates.
- **Overall Performance Improvement through Combined Methods:** By integrating the three directions above—global embeddings, refined evaluation functions, and value gap reduction—our best model (FC-CNN with TD-afterstate +

DDQN TDA-all + TDA-full) achieves a 3-ply score of 5.64×10^5. This performance surpasses previous DNN-based approaches such as CNN22 (3.80×10^5 at 3-ply) and narrows the gap with state-of-the-art n-tuple methods (5.85×10^5 with 6-ply+D). These results demonstrate the effectiveness of enhancing data utilization in bridging the performance gap between DNN-based and traditional 2048 agents.

1.6 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2: Literature Review** – Provides a comprehensive review of relevant literature on data utilization in Deep RL, game AI, and existing approaches to the 2048 game. It discusses the strengths and limitations of current methods and identifies gaps that this research aims to address.
- **Chapter 3: Methodology and Model Design** – Details the proposed methods, focusing on how they enhance data utilization through global embeddings and accurate data generation. It explains the neural network architectures and algorithms developed for this research.
- **Chapter 4: Experimental Setup** – Describes the experimental framework, including the datasets used, training procedures, evaluation metrics, and implementation details. It outlines how the experiments are designed to test the effectiveness of the enhanced data utilization methods.
- **Chapter 5: Results and Discussion** – Presents the experimental results, providing a thorough analysis of how the proposed data utilization methods affect the performance of the Deep RL agents compared to baseline methods.
- **Chapter 6: Conclusion and Future Work** – Summarizes the main contributions of the research, discusses the implications of improved data utilization in Deep RL, addresses the limitations, and offers suggestions for future work.
- **References** – Lists all the academic sources and literature cited throughout the thesis.

1.7 Summary

In summary, this research focuses on enhancing data utilization methods in Deep RL agents for game AI, specifically in the context of the game 2048. By addressing the challenges of effectively utilizing global information, refining evaluation functions, and reducing value gaps, we aim to improve agent performance. The findings from this research have the potential to contribute to the advancement of Deep RL methodologies and provide ideas into effective data utilization strategies across various domains.

Chapter 2

Literature Review

2.1 Introduction

A comprehensive literature review is crucial for situating the current research within the existing body of knowledge and identifying gaps that the study aims to address. In the field of artificial intelligence (AI), particularly in deep reinforcement learning (Deep RL) and game AI, significant advancements have been made over the past decade. Deep RL has demonstrated remarkable capabilities in learning complex behaviors in high-dimensional environments, leading to breakthroughs in areas such as robotics, natural language processing, and strategic game playing [5, 6].

Despite these successes, challenges persist in the efficient utilization of data to enhance learning efficiency and agent performance. In complex and stochastic environments, the way agents collect, process, and leverage data greatly impacts their ability to learn optimal policies [7]. Inefficient data utilization can lead to slow learning, suboptimal performance, and an inability to generalize across different tasks or environments.

In the domain of game AI, Deep RL has been instrumental in achieving superhuman performance in games like Go, Chess, and Shogi through models like AlphaGo and AlphaZero [2, 3]. However, certain games with unique challenges, such as the stochastic tile-placement game 2048, still present difficulties for Deep RL agents in matching or surpassing traditional methods [8, 9].

This literature review explores the fundamental concepts of Deep RL and its integration with data utilization strategies. It examines the importance of effectively leveraging global information and generating accurate training data to improve agent performance. The review also delves into existing AI approaches to the game 2048,

analyzing their strengths and limitations, particularly concerning data utilization and global information processing. By critically assessing current methodologies, this chapter aims to highlight the research gaps that the current study intends to address, setting the foundation for proposing enhanced data utilization methods in Deep RL agents for game AI.

2.2 Deep Reinforcement Learning

Deep reinforcement learning (Deep RL) is a subfield of machine learning that combines reinforcement learning (RL) with deep neural networks. This integration allows agents to learn optimal policies directly from high-dimensional sensory inputs, such as raw images, by approximating value functions or policies using deep learning architectures. Deep RL has achieved remarkable success in various domains, including game playing, robotics, and natural language processing [10].

2.2.1 Fundamentals of Reinforcement Learning

Reinforcement learning is a computational approach to understanding and automating goal-directed learning and decision-making. In RL, an agent interacts with an environment $\mathcal{E}$ over discrete time steps. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi(a_t|s_t)$, receives a scalar reward r_t, and transitions to a new state s_{t+1} [11].

The objective of the agent is to learn a policy that maximizes the expected cumulative reward, also known as the return G_t. The return is typically defined as the discounted sum of future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1},$$

where $\gamma \in [0, 1]$ is the discount factor determining the present value of future rewards [11].

Two central concepts in RL are the state-value function $V^\pi(s)$ and the action-value function $Q^\pi(s, a)$, which estimate the expected return starting from state s and state-action pair (s, a), respectively, under policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s],$$

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a].$$

These value functions satisfy recursive relationships known as the Bellman equations [12], forming the foundation for many RL algorithms.

Reinforcement learning methods can be broadly categorized into:

- **Value-based methods**: These methods focus on estimating the optimal value function. An example is Q-learning [13], which aims to learn the optimal action-value function $Q^*(s, a)$.
- **Policy-based methods**: These methods directly optimize the policy without using a value function. The REINFORCE algorithm [14] is a classic example, utilizing policy gradients to adjust the policy parameters in the direction that maximizes expected rewards.
- **Actor-Critic methods**: These combine value-based and policy-based approaches, where the actor updates the policy based on feedback from the critic, which evaluates the action-value function. An example is the Actor-Critic algorithm proposed by Konda and Tsitsiklis [15].

A fundamental challenge in RL is the trade-off between exploration and exploitation. The agent must balance exploiting known actions that yield high rewards and exploring new actions to discover potentially better strategies [11].

Traditional RL algorithms have been successful in solving problems with small state and action spaces. However, they struggle with high-dimensional inputs, such as raw images or complex sensory data. This limitation has been addressed by integrating deep learning techniques, leading to the development of Deep RL methods capable of handling complex, high-dimensional environments.

2.2.2 Integration of Deep Learning in Reinforcement Learning

The integration of deep learning into reinforcement learning has led to significant advancements, enabling agents to learn effective policies directly from high-dimensional and raw sensory inputs. Deep neural networks serve as powerful function approximators, allowing agents to generalize across large state and action spaces [16].

One of the pioneering works in this area is the Deep Q-Network (DQN) introduced by Mnih et al. [16]. DQN combines Q-learning with convolutional neural networks

(CNNs) to enable agents to learn value functions directly from pixel inputs in Atari 2600 games. Key innovations in DQN include the use of experience replay and target networks to stabilize training.

Policy-based methods have also benefited from deep learning. The Asynchronous Advantage Actor-Critic (A3C) algorithm proposed by Mnih et al. [17] leverages deep neural networks to learn both policy and value functions. A3C utilizes multiple parallel agents to stabilize learning and improve training efficiency.

Deep deterministic policy gradient (DDPG) [18] extends actor-critic methods to continuous action spaces by combining deterministic policy gradients with deep neural networks. Similarly, Proximal Policy Optimization (PPO) [19] introduces a surrogate objective to improve training stability in policy gradient methods.

The integration of deep learning has also enabled the development of algorithms capable of mastering complex games like Go and Chess. The AlphaGo system [20] employs deep neural networks to evaluate board positions and policy moves, combining supervised learning from human expert games with reinforcement learning through self-play.

Overall, the incorporation of deep learning techniques into reinforcement learning frameworks has expanded the applicability of RL to a wide range of challenging domains, making it possible to tackle problems that were previously intractable due to high-dimensional state and action spaces.

2.2.3 Challenges in Deep Reinforcement Learning

Despite the successes of deep reinforcement learning, several challenges remain that hinder its broader adoption and effectiveness.

Sample Inefficiency: Deep RL algorithms often require large amounts of training data to converge to optimal or even satisfactory policies [21]. This sample inefficiency can be problematic in real-world applications where data collection is expensive or time-consuming.

Stability and Convergence Issues: Training deep neural networks within an RL context can be unstable due to issues like catastrophic forgetting, vanishing or exploding gradients, and sensitivity to hyperparameters [22]. The non-stationarity of data distributions in RL, where the data distribution depends on the policy being learned, exacerbates these issues.

Exploration vs. Exploitation Trade-off: Balancing exploration and exploitation is a fundamental challenge in RL [23]. Insufficient exploration can lead to convergence on suboptimal policies, while excessive exploration can slow down learning and increase the risk of encountering harmful states in safety-critical applications.

Credit Assignment Problem: Determining which actions are responsible for future rewards, especially when rewards are sparse or delayed, remains a difficult problem [11].

Overestimation Bias: In value-based methods like Q-learning, overestimation of action values can occur, leading to suboptimal policies [24]. Techniques like Double DQN have been proposed to mitigate this bias.

High Variance in Gradient Estimates: Policy gradient methods can suffer from high variance in gradient estimates, which can slow down learning and require careful tuning of learning rates and batch sizes [25].

Efficient Data Utilization: Efficiently utilizing the collected data to improve learning is crucial. Techniques like experience replay [26] and prioritized experience replay [27] have been introduced, but there is still room for improvement in how agents leverage past experiences.

Safety and Robustness: Ensuring that agents behave safely and robustly in the face of uncertainties and adversarial conditions is an ongoing concern [28].

Addressing these challenges is essential for advancing the field of deep reinforcement learning and enabling its application to a broader range of complex, real-world problems.

2.3 Data Utilization in Deep Reinforcement Learning

Efficient data utilization is a critical aspect of deep reinforcement learning (Deep RL), significantly influencing the learning speed, stability, and overall performance of agents. In Deep RL, agents learn from interactions with the environment, which can be costly in terms of time and computational resources. Therefore, maximizing the value extracted from each piece of data is essential for practical and effective learning [10]. This section discusses the importance of data utilization in Deep RL and reviews techniques developed to enhance it.

2.3.1 Importance of Data Utilization

In reinforcement learning, data efficiency refers to the agent's ability to learn optimal policies from a limited amount of interaction with the environment. Efficient data utilization is crucial for several reasons:

- **Sample Efficiency**: Real-world applications often involve environments where data collection is expensive, time-consuming, or risky. Improving data utilization allows agents to achieve better performance with fewer interactions [29].
- **Learning Stability**: Effective use of data can lead to more stable learning processes by reducing variance and preventing overfitting to recent experiences [27].
- **Performance Improvement**: Maximizing the informational content extracted from each experience helps in faster convergence to optimal policies and improves the agent's ability to generalize across different states and scenarios [21].

In Deep RL, the high dimensionality of state and action spaces amplifies the importance of data utilization. Agents must learn complex mappings from observations to actions, which requires processing vast amounts of data effectively. Inefficient data usage can lead to slow learning rates, increased computational costs, and suboptimal policies.

2.3.2 Techniques for Improving Data Utilization

Several techniques have been developed to enhance data utilization in Deep RL. Key methods include experience replay, prioritized experience replay, and data augmentation strategies.

2.3.2.1 Experience Replay

Experience replay is a technique introduced by Lin [26] to improve data efficiency and break correlations between sequential data samples. In this approach, experiences are stored in a replay buffer and sampled randomly during training. This method allows the agent to reuse past experiences multiple times, improving learning efficiency and stability by smoothing over temporal correlations.

Experience replay has been instrumental in the success of Deep Q-Networks (DQN) [16], where it helped stabilize learning from high-dimensional inputs like raw pixels.

By revisiting past experiences, the agent can continue learning from diverse states and transitions, which is particularly important in environments with sparse rewards.

2.3.2.2 Prioritized Experience Replay

Prioritized experience replay builds upon the basic experience replay mechanism by sampling important experiences more frequently [27]. In standard experience replay, all experiences are sampled uniformly, which may not be optimal since some experiences provide more learning value than others.

In prioritized experience replay, each experience is assigned a priority based on the magnitude of its temporal-difference (TD) error. Experiences with higher TD errors indicate that the agent's predictions were significantly different from the actual outcomes, suggesting that learning from these experiences could reduce the agent's overall error more effectively. By sampling these experiences more frequently, the agent can focus on learning from the most informative data.

This method has been shown to improve learning speed and performance in various Deep RL tasks by ensuring that critical experiences are revisited and learned from more thoroughly [27].

2.3.2.3 Data Augmentation Strategies

Data augmentation involves creating additional training data by applying transformations to existing data without altering their underlying semantics. In Deep RL, data augmentation can help improve generalization and robustness by exposing the agent to a wider variety of inputs [30].

Common data augmentation techniques in RL include random crops, flips, rotations, and noise injection. By training on augmented data, agents can learn more invariant representations and improve performance in environments where observations may vary due to noise or other factors.

For example, Laskin et al. [30] introduced RAD (Reinforcement Learning with Augmented Data), demonstrating that simple data augmentation techniques can significantly improve sample efficiency and performance in visual control tasks.

2.3.2.4 Other Techniques

Additional methods for improving data utilization include:

- **Hindsight Experience Replay (HER)**: Proposed by Andrychowicz et al. [31], HER allows agents to learn from failed attempts by reinterpreting unsuccessful episodes as successful ones for different goals, thus increasing the utility of each experience.

- **Off-Policy Learning**: Algorithms like Soft Actor-Critic (SAC) [32] leverage off-policy data to improve sample efficiency by learning from data generated by different policies.

- **Curiosity-Driven Exploration**: Encouraging agents to explore novel states can lead to the collection of more informative data [33].

These techniques highlight the importance of innovative data utilization strategies in enhancing the performance and efficiency of Deep RL agents. By effectively leveraging available data, agents can learn more robust policies, adapt to complex environments, and achieve better overall performance.

2.3.3 Global Information Integration

Efficient data utilization in deep reinforcement learning (Deep RL) not only depends on how data is collected and stored but also on how it is processed and interpreted by the agent. Integrating global information from the environment plays a crucial role in enhancing data utilization by enabling agents to comprehend and leverage the broader context of their experiences. Traditional Deep RL approaches often emphasize local feature extraction, which can limit the agent's ability to capture comprehensive patterns and dependencies within the data [34, 35]. By incorporating global information, agents can develop more robust and informative representations, leading to improved policy learning and performance.

Enhanced Representation Learning: Integrating global information allows agents to learn representations that encompass the entire state of the environment rather than isolated local features. This holistic understanding facilitates the extraction of more meaningful patterns from the data, enabling the agent to make better-informed decisions [36]. For instance, in complex environments where relationships between different entities are critical, global information integration ensures that these interactions are adequately captured and utilized during the learning process.

Improved Generalization: Agents that effectively integrate global information are better equipped to generalize their learning across diverse states and scenarios. By understanding the global context, agents can apply learned strategies more effectively in new or unseen environments, enhancing their ability to perform well beyond the specific instances encountered during training [37]. This generalization is particularly important in tasks with high-dimensional state spaces, where the variability of states can be substantial.

Efficient Data Utilization: Incorporating global information helps in reducing redundancy and focusing the learning process on the most informative aspects of the data. By emphasizing the relationships and dependencies that span the entire environment, agents can prioritize learning from experiences that provide the most significant insights, thereby making more efficient use of the collected data [38]. This targeted learning approach can lead to faster convergence and improved performance with fewer training samples.

Several techniques have been developed to facilitate the integration of global information into Deep RL agents, each contributing to more effective data utilization:

2.3.3.1 Attention Mechanisms

Attention mechanisms enable agents to selectively focus on relevant parts of their input data by assigning different weights to different elements based on their importance. In the context of Deep RL, attention allows agents to process and prioritize global features that are most pertinent to decision-making [38]. For example, Zambaldi et al. [34] introduced relational deep reinforcement learning, where self-attention mechanisms capture relationships between entities in the environment, enabling agents to reason about interactions at a global scale. This selective focus ensures that the agent utilizes the most informative data, enhancing overall learning efficiency.

2.3.3.2 Graph Neural Networks

Graph Neural Networks (GNNs) provide a framework for modeling relational data, making them well-suited for environments where entities and their interactions form complex structures [39]. In Deep RL, GNNs can encode the global state by representing the environment as a graph, where nodes represent entities and edges represent their relationships [37]. For instance, Wang et al. [37] applied GNNs to multi-agent RL tasks,

allowing agents to learn cooperative strategies by capturing interactions among multiple agents. By modeling these relationships explicitly, GNNs enhance the agent's ability to utilize global information effectively, leading to more informed policy updates.

2.3.3.3 Transformers

Transformers, originally developed for natural language processing, have been adapted for RL tasks to process and integrate global information efficiently [40]. Transformers utilize self-attention mechanisms to model dependencies between different parts of the input data, making them ideal for handling global context in high-dimensional environments. Parisotto and et al. [40] introduced the Gated Transformer-XL architecture for RL, demonstrating improved performance in environments where capturing long-range dependencies is crucial. By leveraging the power of transformers, agents can better utilize global information, enhancing data representation and policy learning.

2.3.3.4 Communication Mechanisms in Multi-Agent RL

In multi-agent reinforcement learning (MARL), effective communication among agents is essential for integrating global information and achieving coordinated behavior [41]. Communication mechanisms enable agents to share information about their observations and intentions, facilitating the aggregation of global data that can inform collective decision-making. Methods such as CommNet [42] and Differentiable Inter-Agent Learning (DIAL) [41] allow for differentiable communication among agents, ensuring that global information is seamlessly incorporated into each agent's policy. This collaborative data utilization enhances the overall performance and efficiency of multi-agent systems.

2.3.3.5 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) decomposes complex tasks into simpler subtasks, enabling agents to operate at multiple levels of abstraction [43]. By structuring learning hierarchically, agents can integrate global goals and context into their decision-making processes. Techniques like the Option-Critic architecture [44] allow agents to learn both low-level actions and high-level strategies, effectively utilizing global information across different temporal and spatial scales. This hierarchical approach enhances

data utilization by ensuring that agents can leverage both local and global information to achieve more effective and efficient learning.

2.3.3.6 World Models

World models involve learning a predictive model of the environment, which agents use to simulate future states and plan actions accordingly [45]. By modeling the global dynamics of the environment, agents can integrate global information into their policy learning, enabling them to make informed decisions based on an understanding of how actions affect the overall state. World models improve sample efficiency by allowing agents to generate synthetic experiences and utilize global information to enhance their learning from real interactions [45].

2.3.4 Summary

Integrating global information into deep reinforcement learning agents is a fundamental strategy for enhancing data utilization. By leveraging techniques such as attention mechanisms, graph neural networks, transformers, communication in multi-agent settings, hierarchical reinforcement learning, and world models, agents can develop more comprehensive and informative representations of their environments. This holistic approach to data processing enables more efficient and effective learning, leading to improved policy performance and greater generalization across complex and high-dimensional tasks. Effective global information integration not only maximizes the utility of collected data but also paves the way for advanced decision-making capabilities in Deep RL agents.

2.4 Game AI and Applications of Deep Reinforcement Learning

2.4.1 Successes of Deep RL in Game AI

Deep reinforcement learning (Deep RL) has achieved remarkable success in the domain of game artificial intelligence (Game AI), demonstrating the ability to learn complex strategies and achieve superhuman performance in various games. This section provides an overview of significant achievements of Deep RL in games such as Atari, Go, Chess, and Shogi.

2.4.1.1 Atari Games

One of the earliest breakthroughs in Deep RL was the development of the Deep Q-Network (DQN) by Mnih et al. [46]. DQN combined Q-learning with deep convolutional neural networks to enable an agent to learn control policies directly from high-dimensional sensory inputs, such as raw pixels, in Atari 2600 games. The agent achieved human-level performance in several games, demonstrating the potential of Deep RL for complex decision-making tasks.

The success of DQN spurred further research, leading to improvements such as Double DQN [47], Dueling Networks [48], and Prioritized Experience Replay [49], which enhanced learning stability and performance.

2.4.1.2 AlphaGo and the Game of Go

The game of Go, with its immense search space and strategic depth, was long considered a grand challenge for AI. Silver et al. [2] introduced AlphaGo, a system that combined deep neural networks with Monte Carlo Tree Search (MCTS) to evaluate board positions and select moves. AlphaGo defeated top professional players, culminating in its victory against the world champion Lee Sedol.

AlphaGo utilized policy networks to predict the probability distribution over legal moves and value networks to estimate the expected outcome from a given position. The system was trained using a combination of supervised learning from expert human games and reinforcement learning through self-play, enabling it to surpass human expertise.

2.4.1.3 AlphaZero: Mastering Chess, Shogi, and Go

Building upon AlphaGo, Silver et al. [50] developed AlphaZero, a more generalized version capable of learning to play Chess, Shogi, and Go without any domain-specific knowledge beyond the basic game rules. AlphaZero used a unified algorithm that combined Deep RL with MCTS, relying solely on self-play to learn optimal strategies.

AlphaZero achieved superhuman performance in all three games, defeating the world-champion programs Stockfish in Chess and elmo in Shogi. This demonstrated the power of Deep RL and self-play in mastering complex games without human expertise.

2.4.1.4 OpenAI Five and Dota 2

In the realm of multiplayer online battle arena (MOBA) games, OpenAI developed OpenAI Five, an AI system that achieved remarkable performance in *Dota 2* [51]. OpenAI Five used a combination of Deep RL techniques and large-scale training to learn policies for controlling multiple agents in a highly complex, dynamic environment.

The system was trained through self-play and was able to defeat professional *Dota 2* teams, showcasing the scalability of Deep RL methods to more intricate and cooperative games.

2.4.1.5 Other Notable Achievements

Deep RL has also been applied successfully to a variety of other games and domains, including:

- **StarCraft II**: DeepMind's AlphaStar achieved Grandmaster level performance in *StarCraft II*, a real-time strategy game with immense complexity [52].
- **Poker**: Libratus and Pluribus, developed by researchers at Carnegie Mellon University, utilized reinforcement learning and search algorithms to defeat top human players in Heads-Up No-Limit Texas Hold'em and multiplayer poker, respectively [53, 54].
- **General Game Playing**: Reinforcement learning agents have demonstrated capabilities in learning to play multiple games using general architectures, highlighting the potential for developing versatile AI systems [55].

These successes underscore the significant advancements in Deep RL and its capacity to tackle complex decision-making tasks in various game environments.

2.4.2 Limitations and Challenges in Game AI

Despite the remarkable achievements, Deep RL faces several limitations and challenges when applied to game AI, particularly in complex or stochastic environments.

2.4.2.1 Sample Inefficiency

Deep RL algorithms often require vast amounts of data to learn effective policies. The sample inefficiency arises from the need for extensive exploration and the high dimensionality of state and action spaces [56]. This can be problematic in environments where data collection is costly or time-consuming.

2.4.2.2 Stability and Convergence Issues

Training Deep RL agents can be unstable due to factors such as function approximation errors, non-stationarity of data distributions, and sensitivity to hyperparameters [57]. Achieving stable convergence to optimal policies remains a significant challenge.

2.4.2.3 Exploration-Exploitation Trade-off

Balancing exploration and exploitation is critical in Deep RL. Insufficient exploration can lead to suboptimal policies, while excessive exploration may result in inefficient learning or unsafe behaviors [58]. Designing exploration strategies that are both efficient and effective is an ongoing area of research.

2.4.2.4 Generalization and Transfer Learning

Deep RL agents often struggle to generalize learned policies to new environments or variations of the same task [59]. The lack of transferability limits the applicability of agents trained in specific settings to broader contexts.

2.4.2.5 Complex and Stochastic Environments

In games with high degrees of randomness or partial observability, such as 2048 or procedurally generated environments, Deep RL agents may perform poorly [60]. The stochastic nature of these games introduces additional challenges in learning stable and robust policies.

2.4.2.6 Computational Resources and Scalability

Many successful Deep RL applications, such as AlphaGo and OpenAI Five, require substantial computational resources and specialized hardware [2, 51]. This raises concerns about the scalability and accessibility of such methods for wider use.

2.4.2.7 Ethical and Safety Considerations

As AI agents become more capable, ensuring their safe and ethical deployment becomes increasingly important. Issues such as unintended behaviors, exploitation of game mechanics, or alignment with human values present challenges that need to be addressed [61].

2.4.3 Areas for Improvement

Addressing these limitations requires advancements in several areas:

- **Data Efficiency**: Developing algorithms that can learn effectively from fewer interactions, such as model-based RL or leveraging prior knowledge [62].
- **Stability and Robustness**: Enhancing training procedures and architectures to improve stability and convergence [63].
- **Exploration Strategies**: Designing smarter exploration methods that balance the trade-off more effectively [64].
- **Generalization Techniques**: Implementing approaches that promote better generalization and transfer learning, such as meta-learning or representation learning [65].
- **Scalable Algorithms**: Developing more computationally efficient algorithms that can achieve high performance without excessive resource requirements [66].
- **Safety and Ethics**: Incorporating safety constraints and ethical considerations into the design of AI agents [67].

By addressing these challenges, the field can advance towards creating more capable, efficient, and reliable AI agents for complex and stochastic game environments.

2.5 The Game 2048 and Artificial Intelligence Approaches

2.5.1 Overview of the 2048 Game

The game 2048 is a single-player sliding block puzzle game developed by Gabriele Cirulli in 2014 [68]. It is played on a 4×4 grid with numbered tiles that slide when a player moves them using the four directional arrows: up, down, left, or right. When two tiles

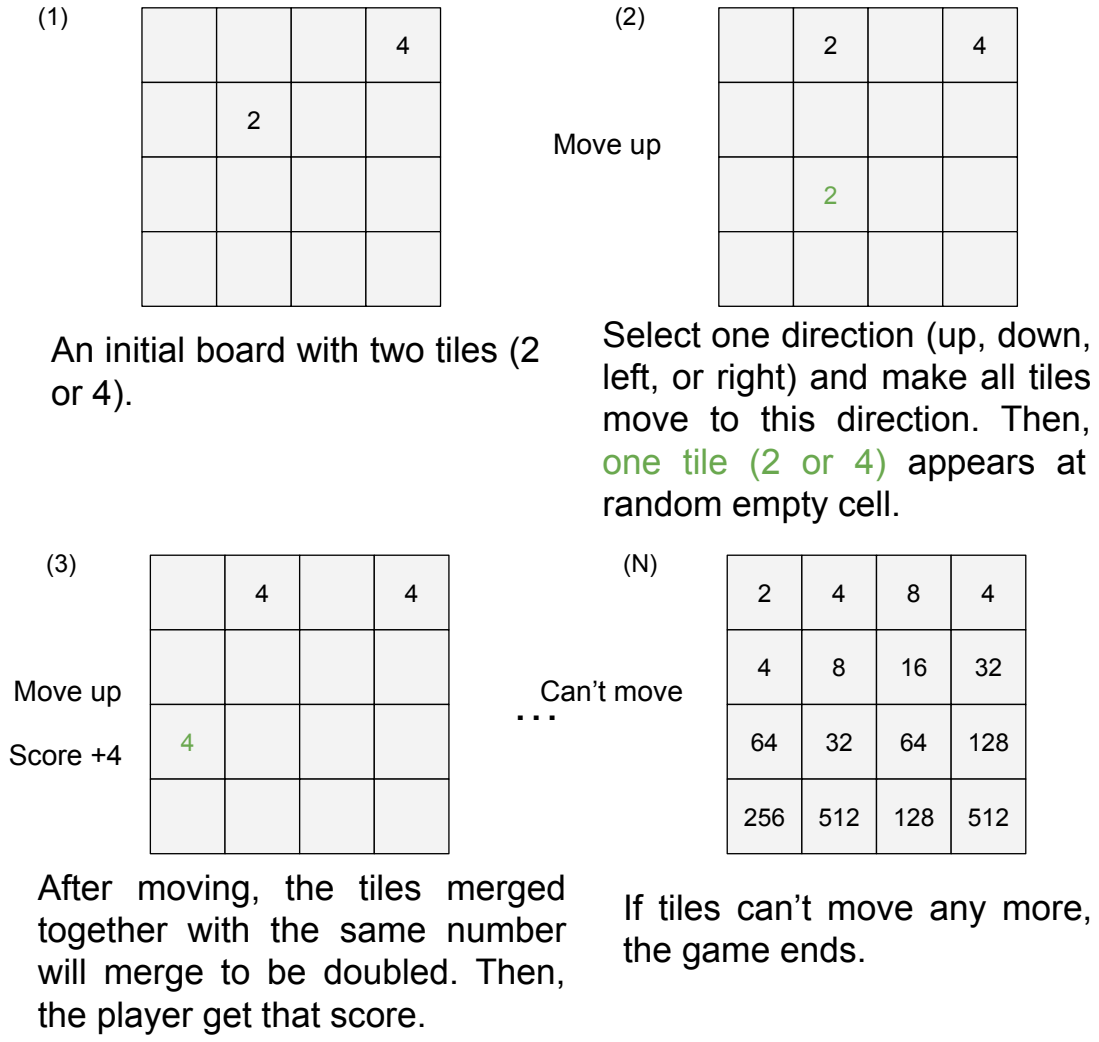


Figure 2.1: An example of the game 2048.

with the same number collide during a move, they merge into a tile with the sum of their numbers. The objective is to create a tile with the number 2048, although players can continue playing to reach higher numbers.

At the start of the game, two tiles appear in random positions on the grid, each with a value of either 2 or 4. After each move, a new tile with a value of 2 (90% probability) or 4 (10% probability) spawns in a random empty cell [69]. The stochastic nature of tile placements introduces randomness into the game, making it challenging to predict future states and plan accordingly.

The game ends when no valid moves are possible, which occurs when the grid is full, and no adjacent tiles have the same value. The player's score increases by the value of the tiles combined during moves, incentivizing the creation of higher-value tiles. The

simplicity of the game's rules contrasts with its strategic depth, making it an interesting subject for artificial intelligence research. Figure 2.1 shows an example of the game 2048.

2.5.2 Traditional AI Approaches to 2048

Several traditional AI techniques have been applied to the game 2048, achieving notable success in reaching high scores. Two prominent methods are expectimax search algorithms and N-tuple networks.

2.5.2.1 Expectimax Search Algorithms

Expectimax is an extension of the minimax algorithm used in decision-making processes, adapted to handle stochastic environments by incorporating chance nodes into the game tree [70]. In the context of 2048, expectimax search considers both the player's moves and the random tile placements as part of the game tree, evaluating the expected utility of each action by averaging over possible chance outcomes.

Implementations of expectimax for 2048, such as the one by Szubert and Jaśkowski [69], have demonstrated strong performance by exploring possible future states to a certain depth and selecting moves that maximize the expected score. The primary advantage of expectimax is its ability to plan ahead by considering the probabilistic outcomes of tile placements, making it effective in managing the game's stochasticity.

However, the expectimax approach faces limitations due to the exponential growth of the game tree with increased search depth, leading to high computational costs. Practical implementations often limit the search depth to manage computational resources, which can restrict the algorithm's ability to anticipate long-term consequences of actions fully.

2.5.2.2 N-Tuple Networks

N-tuple networks are value function approximators that estimate the value of a game state based on patterns (tuples) extracted from the board [71]. In 2048, an N-tuple network consists of several tuples, each representing a specific arrangement of tiles, and associates weights with these tuples to approximate the expected utility of the game state.

Szubert and Jaśkowski [69] applied temporal difference learning with N-tuple networks to 2048, achieving state-of-the-art performance at the time. The method involves updating the weights of the tuples based on the difference between predicted and actual rewards, allowing the network to learn effective value estimations from experience.

The strengths of N-tuple networks include their low computational requirements and the ability to capture important features of the game through carefully designed tuples. They can generalize across similar states and are efficient for online learning. However, their performance heavily depends on the selection of tuples and the quality of the learning process. Designing effective tuples requires domain knowledge, and the networks may struggle to capture long-range dependencies and global board configurations due to their focus on local patterns.

2.5.3 Deep Reinforcement Learning Approaches to 2048

Deep reinforcement learning (Deep RL) methods have been explored for 2048 to leverage deep neural networks' ability to handle high-dimensional state spaces. These approaches aim to learn value functions or policies directly from raw game states without the need for manual feature engineering.

One common approach involves using deep Q-networks (DQN) [46], where a convolutional neural network (CNN) processes the game board and outputs Q-values for each possible action. The agent learns by minimizing the temporal difference error between predicted and target Q-values, using techniques like experience replay to stabilize training.

Despite the potential of Deep RL, applying these methods to 2048 has not surpassed the performance of traditional approaches like N-tuple networks [69]. Several challenges contribute to this outcome:

- **Limited Utilization of Global Information**: CNNs primarily focus on local feature extraction due to their limited receptive fields. This constraint hinders the network's ability to capture global board configurations and long-range dependencies essential for strategic planning in 2048.
- **High Variance in Training Data**: The stochastic tile placements introduce significant randomness, leading to high variance in the training data. This makes it difficult for neural networks to learn accurate value estimations and can cause instability during training.

- **Shallow Search Depths**: Deep RL methods often rely on single-step updates without incorporating deeper search strategies. This limitation restricts the agent's foresight and ability to plan ahead effectively, which is crucial for achieving higher scores in the game.

- **Low Exploration**: Existing Deep RL methods in 2048 often update only the best afterstate during training, which limits exploration of alternative possibilities. This problem leads to underutilization of valuable information from non-best afterstates, causing value gaps between 1-ply and 2-ply evaluations and reducing the stability of learning.

Efforts to improve Deep RL performance in 2048 have included experimenting with different neural network architectures and training techniques. However, these methods often come with increased computational complexity and have yet to match the effectiveness of traditional approaches. The challenges show the need for novel methods that enhance data utilization and integrate global information into the learning process.

2.6 Summary and Research Gap Identification

This literature review has explored the fundamentals of deep reinforcement learning, the significance of efficient data utilization, and the application of AI methods to the game 2048. Traditional AI approaches, such as expectimax search and N-tuple networks, have achieved considerable success but come with limitations, including high computational costs and reliance on manual feature engineering.

Deep RL methods, on the other hand, offer the potential to learn directly from raw game states without manual feature design. However, in the context of 2048, these methods have yet to surpass traditional approaches due to challenges such as inefficient data utilization, high variance in training data, limited capacity to process global information, shallow planning depth, and insufficient exploration.

The key research gaps identified are summarized as follows:

- **Enhancing Data Utilization:** Existing Deep RL methods often fail to fully exploit available gameplay data. Reducing variance in training and improving the stability of learning are essential to achieving stronger performance.

- **Integrating Global Information:** Current architectures, such as CNN-based models, primarily capture local patterns but lack the ability to effectively process

global board configurations, which are crucial for long-term strategic decision-making in 2048.

- **Improving Planning and Foresight:** Many Deep RL approaches rely on shallow one-step updates, limiting their ability to anticipate future consequences. Incorporating deeper search strategies or refining evaluation functions is necessary to enhance decision quality.

- **Strengthening Exploration:** Standard methods tend to update only the best afterstate during training, resulting in under-exploration of alternative states and large value gaps between 1-ply and 2-ply evaluations.

Addressing these research gaps is essential for advancing the performance of Deep RL agents in 2048 and similar stochastic, high-dimensional environments. This motivates the present study, which proposes three directions to enhance data utilization: (1) leveraging global embedding techniques to better capture holistic information, (2) refining evaluation functions to reduce variance and improve training stability, and (3) reducing value gaps by enhancing exploration ability. These strategies together aim to push the boundaries of Deep RL in 2048 and bridge the performance gap between Deep RL agents and traditional methods.

Chapter 3

Methodology and Model Design

3.1 Global Embedding

In this section, we present enhancements to the neural network architecture. These enhancements involve integrating global information embeddings into our baseline neural network, CNN22. The primary objective of this improvement is to augment the network's ability to comprehend and utilize global contextual information. This is done alongside local feature extraction. By embedding global information, we enable the neural network to make more informed and strategic decisions, thereby enhancing its performance in the target game, 2048. This integration involves generating a global embedding vector that encapsulates the overall state of the game board and incorporating this vector into the existing convolutional layers of the baseline CNN22. The fusion of global and local features facilitates a more holistic understanding of the game environment, leading to improved decision-making and higher efficiency in learning optimal strategies.

3.1.1 Overview

3.1.1.1 Research Background

Deep Reinforcement Learning (Deep RL) has emerged as a powerful approach in the field of Game AI, enabling agents to learn optimal strategies through interaction with complex environments. By leveraging deep neural networks, Deep RL can effectively handle high-dimensional state spaces and make sequential decisions, which are essential for mastering various games. Notable successes include DeepMind's AlphaGo and AlphaZero, which

have demonstrated superhuman performance in strategic games such as Go, Chess, and Shogi [2, 3].

In the context of the 2048 game, Deep RL offers promising methods for developing intelligent agents capable of achieving high scores. However, prior to this work, the best open sourced DNN-based 2048 agents [72] have not outperformed the state-of-the-art agent based on N-Tuple Networks [4]. Given the success of DNNs in other games, we expect that it is still possible to improve their performance in 2048 by addressing these limitations, potentially through enhanced feature representation and better strategy formulation mechanisms.

3.1.1.2 Areas for Improvement

Our baseline model, CNN22, is structured as shown in Figure 3.1. CNN22 consists of two convolutional layers and three fully connected (FC) layers. The input to CNN22 is a one-hot encoding representation of the board's values, and the output is the valuation of the current afterstate, representing the expected cumulative reward from the current afterstate until the end of the game. While the FC layers enable CNN22 to capture and analyze global information, the shallow convolutional layers are limited to processing only local information due to the inherent constraints of convolutional networks.

The inherent constraints of convolutional networks primarily arise from their localized receptive fields and the nature of the convolution operation. Mathematically, a convolutional layer applies a set of filters (kernels) that perform element-wise multiplication and summation over local regions of the input data. Each filter typically has a fixed size, such as 2×2, defining the extent of the local neighborhood it can process. In a shallow network with only a few convolutional layers, the effective receptive field—the region of the input data that influences a particular output unit—is relatively small. For instance, with two consecutive 2×2 convolutional layers, the receptive field expands to 3×3, allowing the network to capture patterns within this localized area. However, global contextual information, which requires understanding relationships and dependencies across the entire board, remains inaccessible because the network cannot aggregate information from distant regions without significantly increasing the depth or employing techniques like dilated convolutions or pooling layers. Addressing the mathematical limitation by incorporating mechanisms that facilitate the integration of global

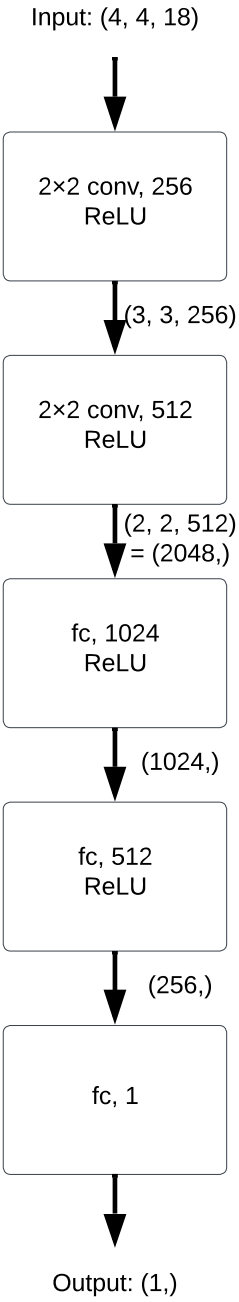


Figure 3.1: The structure of CNN22.

information into the convolutional layers may be essential for enhancing the performance of CNN22.

Global information is highly beneficial for strategic decision-making in the 2048 game. For example, knowing the position of the maximum tile can guide the placement and merging of other tiles to prevent high-value tiles from being blocked. Additionally, understanding the complete state of each row and column allows the network to make more accurate evaluations of potential moves, ensuring that merges are optimized for

future steps. Therefore, providing global contextual information to the shallow convolutional layers presents a significant opportunity for enhancing the performance of CNN22. By integrating global embeddings directly into the convolutional layers, the network can better understand and utilize comprehensive game state information, leading to more informed and effective decision-making processes.

3.1.1.3 Core Concepts

Our core concept revolves around integrating global board information into the shallow convolutional layers of the baseline CNN22 model. By doing so, we aim to enhance the network's ability to capture comprehensive game state information. This integration is achieved through the introduction of global embedding vectors that encapsulate the overall board state and are merged with the original input of our baseline neural network. Mathematically, this involves concatenating the global embedding vectors to the inputs of the convolutional layers, thereby providing the network with both local and global perspectives of the game state. For instance, knowing the position of the maximum tile and the state of each row and column may improve the accuracy of afterstate valuations by informing the network about potential merging opportunities and strategic tile placements. This holistic approach is expected to improve the accuracy of afterstate valuations and enable the agent to develop more effective strategies for achieving higher scores.

3.1.2 Model and Algorithm Design

In this subsection, we delve into the design and innovative aspects of the four proposed global embedding methods: BP-CNN, FC-CNN, CNN-CNN, and T-CNN. Each method offers a unique approach to integrating global contextual information into the baseline CNN22 model, thereby enhancing its performance in the 2048 game.

3.1.2.1 Bypass CNN (BP-CNN)

Design Philosophy BP-CNN introduces a straightforward yet effective modification to the baseline CNN22 by incorporating positional information directly into the input layer. In the original CNN22 architecture, the input is a one-hot encoded tensor of size $4 \times 4 \times 18$, where each tile's value is represented by an 18-dimensional one-hot vector.

This encoding effectively captures the value of each tile but lacks explicit information about the tile's position on the board.

To address this limitation, BP-CNN extends the input representation by concatenating one-hot encoded vectors of the x and y coordinates to each tile's value vector. Specifically, for each tile, an additional one-hot vector representing its x-coordinate and another for its y-coordinate are appended, resulting in a new input tensor of size $4 \times 4 \times 26$. Mathematically, if $V_{i,j}$ denotes the 18-dimensional one-hot vector for the tile at position (i, j), X_i represents the one-hot encoded x-coordinate, and Y_j represents the one-hot encoded y-coordinate, then the BP-CNN input $I_{i,j}$ for each tile is given by:

$$I_{i,j} = [V_{i,j}; X_i; Y_j]$$

where $[;]$ denotes the concatenation operation.

The neural network structure of BP-CNN is shown in Figure 3.2.

Innovation and Benefits The primary innovation of BP-CNN lies in its ability to provide the neural network with explicit spatial context. By informing the network of each tile's exact location, BP-CNN enhances the model's capacity to recognize and utilize spatial patterns critical for strategic decision-making in the 2048 game. For instance, tiles positioned at the edges or corners of the board hold different strategic values compared to those in other positions. Understanding the value of tiles in these specific locations can help prevent high-value tiles from being blocked and enable more effective merging strategies.

This modification offers several advantages:

- **Enhanced Feature Representation:** By combining value and positional data, the network gains a more comprehensive understanding of the game state, allowing for more informed decision-making.
- **Minimal Architectural Changes:** BP-CNN achieves these enhancements without increasing the network's depth or complexity, maintaining computational efficiency while significantly boosting performance.

In summary, BP-CNN leverages the straightforward strategy of augmenting the input representation with positional information. This modification not only enhances the network's ability to process comprehensive game state information but also lays the

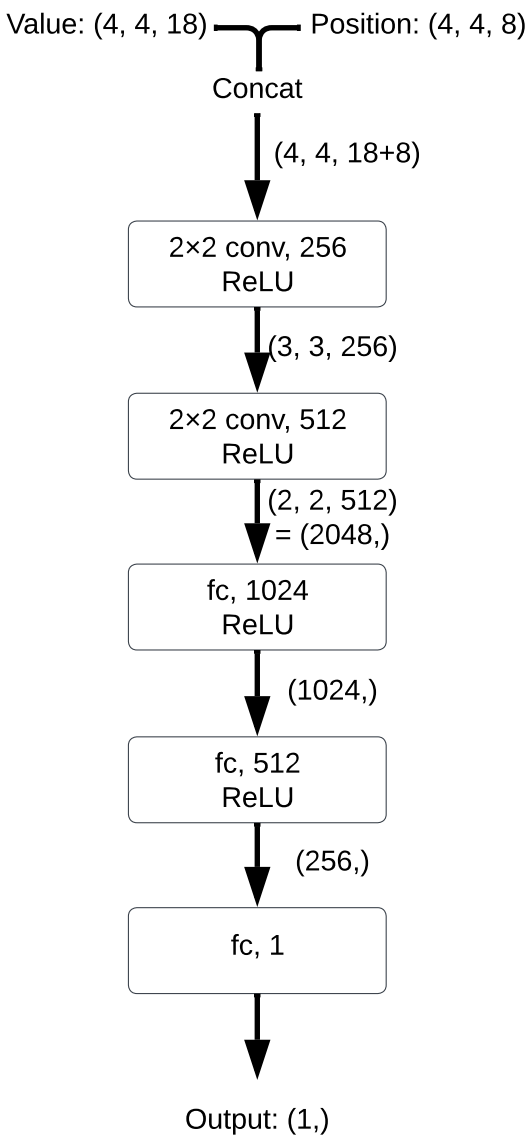


Figure 3.2: The structure of BP-CNN.

groundwork for further architectural innovations aimed at optimizing strategic decision-making in the 2048 game.

3.1.2.2 Fully Connected Embedding CNN (FC-CNN)

Fully Connected Embedding CNN (FC-CNN) enhances the baseline CNN22 by integrating global contextual information through dedicated fully connected (FC) layers.

Building upon the BP-CNN approach, FC-CNN processes the enriched input tensor of size $4 \times 4 \times 26$ and embeds global information more effectively.

Design Philosophy

In the original CNN22 architecture, the input is a one-hot encoded tensor of size $4 \times 4 \times 18$, representing the values of each tile on the 2048 game board. BP-CNN extends this input by concatenating one-hot encoded x and y coordinates to each tile's value vector, resulting in an input tensor of size $4 \times 4 \times 26$. FC-CNN further leverages this enriched input by introducing fully connected layers to create a robust global embedding.

Specifically, the embedding layers of FC-CNN are detailed as follows:

- **Input Reshaping and Flattening:** The input tensor V_{input} of size $(4, 4, 26)$ is reshaped and flattened into a vector V_{flat} of size $(416,)$, where $416 = 4 \times 4 \times 26$.
- **Global Embedding via Fully Connected Layers:** The flattened input V_{flat} is processed through two fully connected layers to embed global information:

$$F_1 = \text{ReLU}(W_1 V_{\text{flat}} + b_1), \quad W_1 \in \mathbb{R}^{32 \times 416},$$
$$F_2 = \text{ReLU}(W_2 F_1 + b_2), \quad W_2 \in \mathbb{R}^{128 \times 32},$$

  where F_1 is a 32-dimensional vector, and F_2 is a 128-dimensional global embedding.
- **Reshaping the Embedding:** The global embedding F_2 is reshaped into a tensor E of size $(4, 4, 8)$, as $4 \times 4 \times 8 = 128$.
- **Extracting Value Channels and Concatenation:** The original value one-hot encoded input V_{value} of size $(4, 4, 18)$ is extracted from V_{input}. The reshaped embedding E is then concatenated with V_{value} along the channel dimension to form a new input tensor I of size $(4, 4, 26)$:

$$I = [V_{\text{value}}; E],$$

  where $[\,;\,]$ denotes concatenation along the channel axis.

The neural network structure is shown as the figure 3.3

Innovation and Benefits

The primary innovation of FC-CNN lies in its method of embedding global information through fully connected layers, enabling the network to capture and integrate

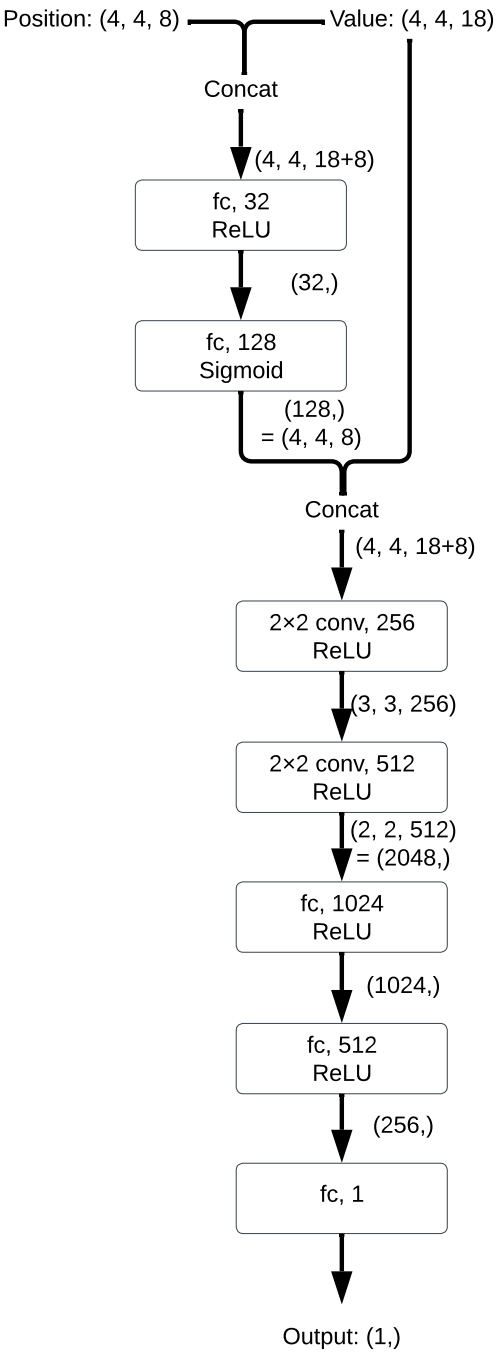


Figure 3.3: The structure of FC-CNN with global embedding.

comprehensive game state information more effectively. By embedding global context directly into the input tensor, FC-CNN provides the convolutional layers with enriched feature representations, allowing for more accurate valuations of afterstates and improved strategic decision-making in the 2048 game.

This approach offers several advantages:

- **Enhanced Global Feature Integration:** The fully connected layers effectively capture and embed global patterns, providing the network with a holistic view of the game state.
- **Improved Decision-Making Accuracy:** With comprehensive global information, the network can make more informed and strategic moves, leading to higher performance in the game.
- **Combining FC and CNN Strengths:** The architecture merges the global context modeling of fully connected layers with the local feature extraction capabilities of convolutional layers.
- **Inherited Positional Context from BP-CNN:** By utilizing the enriched input from BP-CNN, FC-CNN builds upon existing positional information, enhancing the network's understanding of tile placement and strategic importance.

Strategic Importance

Global information embedded by the fully connected layers is crucial for strategic decision-making in the 2048 game. FC-CNN can identify patterns and relationships involving the overall board configuration, such as the distribution of high-value tiles and potential merging paths. By processing the entire board's state through fully connected layers, the network gains a comprehensive understanding of the game state, helping to avoid unfavorable moves and promoting strategies that may lead to higher scores.

Summary

FC-CNN addresses the limitations of the baseline CNN22 by embedding global information through fully connected layers. This enhancement allows the network to better understand and utilize comprehensive game state information, leading to more informed and effective strategic decision-making in the 2048 game.

3.1.2.3 Convolutional Embedding CNN (CNN-CNN)

Convolutional Embedding CNN (CNN-CNN) enhances the baseline CNN22 by integrating global contextual information through additional convolutional layers dedicated to embedding. Building upon the BP-CNN approach, CNN-CNN processes the enriched input tensor of size $4 \times 4 \times 26$ and embeds global information more effectively.

Design Philosophy

In the original CNN22 architecture, the input is a one-hot encoded tensor of size $4 \times 4 \times 18$, representing the values of each tile on the 2048 game board. BP-CNN extends this input by concatenating one-hot encoded x and y coordinates to each tile's value vector, resulting in an input tensor of size $4 \times 4 \times 26$. CNN-CNN further leverages this enriched input by introducing convolutional layers to create a robust global embedding.

The embedding layers of CNN-CNN are detailed as follows:

- **Global Embedding via Convolutional Layers:** The input V_{input} is processed through two convolutional layers to embed global information:
  - **First Convolutional Layer (cc1):** Applies 16 filters of size 2×2 with stride 1 and no padding, producing an output tensor C_1 of shape $(3, 3, 16)$:

$$C_1 = \text{ReLU}(\text{Conv}_{cc1}(V_{\text{input}}))$$

  - **Second Convolutional Layer (cc2):** Applies 128 filters of size 3×3 with stride 1 and no padding, resulting in an output tensor C_2 of shape $(1, 1, 128)$:

$$C_2 = \text{ReLU}(\text{Conv}_{cc2}(C_1))$$

- **Reshaping the Embedding:** The output C_2 is reshaped into a tensor E of size $(4, 4, 8)$ to align with the spatial dimensions of the original input, as $4 \times 4 \times 8 = 128$.
- **Extracting Value Channels and Concatenation:** The original value one-hot encoded input V_{value} of size $(4, 4, 18)$ is extracted from V_{input}. The reshaped embedding E is then concatenated with V_{value} along the channel dimension to form a new input tensor I of size $(4, 4, 26)$:

$$I = [V_{\text{value}}; E],$$

  where $[\,;\,]$ denotes concatenation along the channel axis.

The structure of CNN-CNN is shown as the following figure 3.4

Innovation and Benefits

The primary innovation of CNN-CNN lies in its method of embedding global information through dedicated convolutional layers, leveraging the inherent strengths of

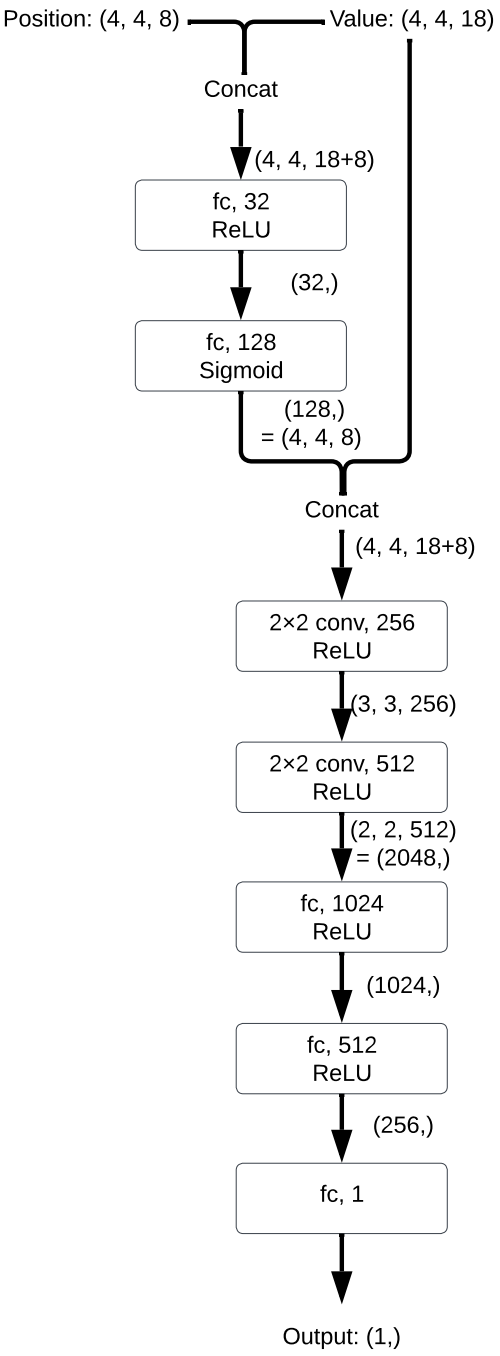


Figure 3.4: The structure of CNN-CNN with global embedding.

convolutional operations to capture comprehensive game state information. By embedding global context directly into the input tensor via convolutional layers, CNN-CNN provides the convolutional layers with enriched feature representations, allowing for more accurate valuations of afterstates and improved strategic decision-making in the 2048

game.

This approach offers the following advantages:

- **Enhanced Global Feature Integration:** The two additional convolutional layers effectively capture and embed global patterns, providing the network with a holistic view of the game state.
- **Improved Decision-Making Accuracy:** With comprehensive global information, the network can make more informed and strategic moves, leading to higher performance in the game.
- **Leveraging Spatial Hierarchies:** Convolutional layers inherently capture spatial hierarchies and patterns, enhancing the network's ability to recognize complex relationships across the board.
- **Inherited Positional Context from BP-CNN:** By utilizing the enriched input from BP-CNN, CNN-CNN builds upon existing positional information, further enhancing the network's understanding of tile placement and strategic importance.

Strategic Importance

Global information is highly beneficial for strategic decision-making in the 2048 game. Unlike FC-CNN, which primarily focuses on embedding positional information, CNN-CNN leverages convolutional layers to capture complex spatial dependencies and global patterns across the entire board. When capturing image features or board features, CNNs may be more effective than fully connected layers.

Summary

CNN-CNN addresses the inherent limitations of shallow convolutional layers by embedding global information through additional convolutional layers. This enhancement allows the network to better understand and utilize comprehensive game state information, leading to more informed and effective strategic decision-making in the 2048 game.

3.1.2.4 Transformer Embedding CNN (T-CNN)

T-CNN enhances the baseline CNN22 by integrating global contextual information through a Transformer Encoder layer. Building upon the previous models, T-CNN utilizes the Transformer architecture to capture complex global dependencies across the entire board, providing a powerful mechanism for embedding global information.

Design Philosophy

In the original CNN22 architecture, the input is a one-hot encoded tensor of size $4 \times 4 \times 18$, representing the values of each tile on the 2048 game board. BP-CNN and FC-CNN extend this by incorporating positional information and embedding global context through fully connected layers or additional convolutional layers. T-CNN introduces a Transformer Encoder layer to process the enriched input, capturing global relationships more effectively due to the self-attention mechanism inherent in Transformers.

Specifically, T-CNN processes the input tensor as follows:

1. **Input Reshaping and Preparation:** The input tensor V_{Input} of size $4 \times 4 \times 26$ (including positional information from BP-CNN) is reshaped to a sequence suitable for the Transformer Encoder. It is reshaped to $(16, 26)$, where each of the 16 positions corresponds to a tile on the board.

2. **Transformer Encoding:** A Transformer Encoder layer processes the input sequence, capturing global dependencies through self-attention. The output E_{Trans} retains the same shape $(16, 26)$.

3. **Global Embedding Compression:** The output from the Transformer Encoder is passed through a linear layer to compress the embedding from 26 dimensions to 8 dimensions, resulting in E_{Comp} of shape $(8, 16)$.

4. **Reshaping and Concatenation:** The compressed embedding E_{Comp} is reshaped back to spatial dimensions $(4, 4, 8)$. It is then concatenated with the original input of size $(4, 4, 18)$, forming a new input tensor I of size $(4, 4, 26)$:

$$I = [V_{Input}; E_{Comp}^{reshaped}]$$

where [;] denotes concatenation along the channel dimension.

The overall architecture effectively combines the strengths of Transformer models in capturing global dependencies with the spatial feature extraction capabilities of convolutional layers. The structure is shown as the Figure 3.5.

Innovation and Benefits

The primary innovation of T-CNN lies in its utilization of a Transformer Encoder layer to embed global information, leveraging the self-attention mechanism to capture

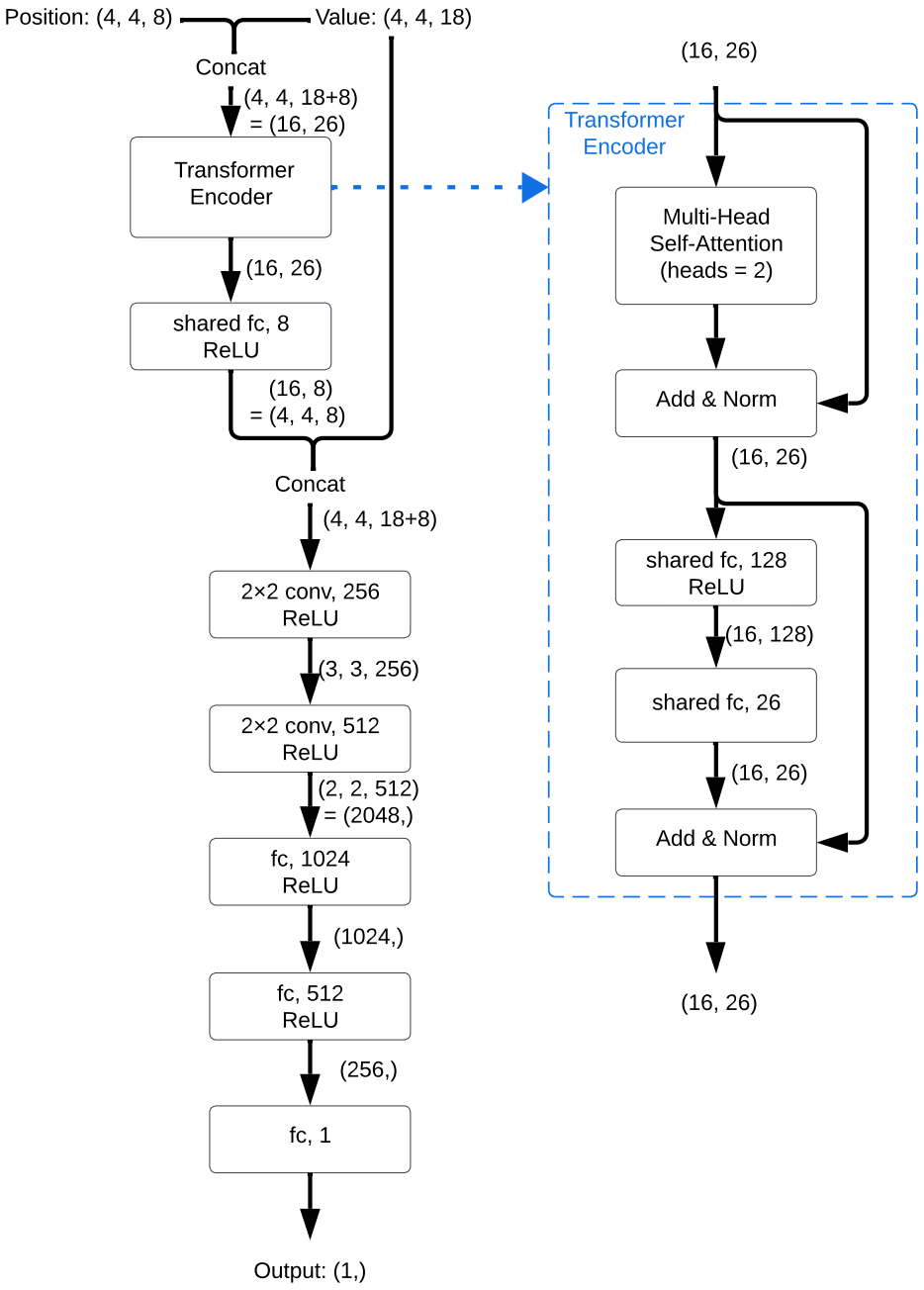


Figure 3.5: The structure of T-CNN with global embedding. The structure of the Transformer Encoder is the same as the paper [1]

complex relationships across the entire game board. This approach offers several advantages:

- **Advanced Global Feature Integration:** The Transformer Encoder effectively models long-range dependencies and global interactions between tiles, providing a comprehensive understanding of the game state.

- **Enhanced Decision-Making Accuracy:** By capturing intricate global patterns, T-CNN enables the network to make more informed and strategic moves, leading to improved performance in the game.
- **Combining Transformer and CNN Strengths:** The architecture merges the global context modeling of Transformers with the local feature extraction of convolutional layers, optimizing the network's ability to process both global and local information.
- **Scalability and Flexibility:** The Transformer component allows for flexible modeling of input sequences of varying lengths and can be extended or adjusted to capture additional complexities in the game state.

Strategic Importance

Global information captured by the Transformer Encoder is crucial for strategic decision-making in the 2048 game. T-CNN can identify complex patterns and relationships that are not easily captured by convolutional layers alone. For instance, it can detect potential merging opportunities that involve tiles located far apart on the board or recognize patterns that could lead to deadlocks if not addressed promptly. By modeling these global dependencies, T-CNN provides more accurate evaluations of afterstates, helping to prevent unfavorable moves and promoting strategies that lead to higher scores.

Summary

T-CNN addresses the limitations of the baseline models by integrating a Transformer Encoder layer to embed global information effectively. This enhancement allows the network to better understand and utilize comprehensive game state information, combining the advantages of Transformers and convolutional neural networks. As a result, T-CNN may improve strategic decision-making in the 2048 game.

3.1.3 Summary of Global Embedding

This section has detailed the development and theoretical underpinnings of the global embedding technique within the broader scope of enhancing decision-making in game AI, specifically for the game 2048. By integrating global contextual information into the neural network architecture, the global embedding approach is designed to leverage a

comprehensive understanding of the game state, thereby facilitating more informed and strategic decision-making processes.

Technical Contributions: The primary innovation of the global embedding technique lies in its ability to synthesize and incorporate broad game state data within the network. This is achieved through the novel application of embedding layers that process not just local tile information but also the relational and positional context of the game board. These layers transform the raw data into a format that is more conducive to extracting strategic insights, which may be crucial for high-level game play in 2048.

Theoretical Implications: Theoretically, this approach extends the traditional use of convolutional neural networks in game AI by adding a layer of abstraction that captures global dynamics, which are often missed by purely local analyses. This method promises to overcome some of the common limitations encountered in traditional game AI architectures, such as the inability of shallow network layers in standard CNNs to fully capture the comprehensive information of the game board. By incorporating global embedding, the network can gain the capability to access and utilize a complete representation of the game state at every decision point, enhancing its strategic accuracy.

Potential Applications: The global embedding technique developed in this research has broad applicability beyond the specific case of the 2048 game. By providing a comprehensive view of the entire game board at any given decision point, this method can be adapted to other strategic games that require a deep understanding of current game states. Examples include chess or Go, where the ability to evaluate the entire board configuration in real-time can fundamentally enhance decision-making processes. Moreover, this approach can also be beneficial in non-gaming contexts that demand a holistic understanding of complex scenarios. For instance, in robotics, where navigating an environment effectively requires a complete and instant understanding of surrounding elements, or in traffic management systems, where the state of an entire network needs to be considered to optimize flow and reduce congestion. By enabling systems to integrate and analyze all relevant data points simultaneously, global embedding can significantly improve the effectiveness of systems in various fields that rely on comprehensive situational awareness.

Future Directions: Future research will focus on refining the efficiency of the embedding process to reduce computational demands and enhance real-time applicability. Additionally, exploring adaptive and dynamic embedding techniques that can adjust to the evolving context of the game or task at hand will be critical. Continued development

will also aim to validate and quantify the benefits of global embedding through empirical testing, which will be crucial for substantiating the theoretical advantages proposed.

In conclusion, the global embedding framework sets a foundational methodology for enhancing neural network architectures with a global perspective, promising significant advancements in the field of game AI and beyond.

3.2 Refining Evaluation Functions

To enhance the performance of neural network-based agents in the game 2048, it is crucial to focus on the generation and utilization of more accurate training data. Traditional methods, while effective to a certain extent, are often hindered by high variance and shallow search strategies, which limit the agent's ability to make well-informed decisions. In this section, we introduce our advanced training methodologies designed to address these limitations. By generating data with reduced randomness and incorporating deeper search techniques, our methods aim to provide a more stable and comprehensive training framework.

3.2.1 Overview

This section introduces our approach to enhancing the training of 2048 game agents by generating and utilizing more accurate data. We begin with the research background, highlighting the use of Temporal Difference (TD) learning in previous works and our baseline model. We then identify areas for improvement in existing methods and present the core concepts of our proposed techniques.

3.2.1.1 Research Background

Temporal Difference (TD) learning [73] is a fundamental reinforcement learning method that combines ideas from Monte Carlo methods and dynamic programming. It allows agents to learn directly from raw experience. TD learning has been successfully applied in various domains, including game playing.

In the context of the 2048 game, prior to this work, the best open-sourced deep neural network (DNN)-based agents [72] employed the TD-afterstate method for training. TD-afterstate is a variant of TD(0). This approach enables agents to learn value functions that evaluate afterstates and guide decision-making effectively. The updating

method for TD-afterstate is shown as the Fig. 3.6. While TD-afterstate is effective, this method has limitations that present opportunities for improvement.

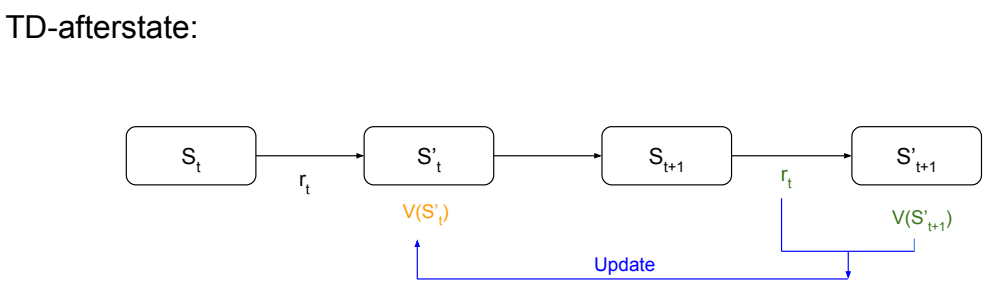


Figure 3.6: The updating method of TD-afterstate

3.2.1.2 Areas for Improvement

Despite the success of TD-afterstate in training 2048 agents, there are notable challenges that hinder optimal performance:

- **High Variance in Training Data:** The randomness inherent in the 2048 game, particularly the random placement of new tiles after each move, introduces significant variance in the training data. This variance can lead to less stable learning and suboptimal estimation of state values.
- **Shallow Search Depth:** Traditional TD(0) methods rely on updates based on immediate rewards and the estimated value of the next state. This shallow search may not capture the long-term consequences of actions, limiting the agent's ability to plan ahead effectively.

Addressing these issues is crucial for developing agents that can achieve higher scores and exhibit more strategic play in the 2048 game.

3.2.1.3 Core Concepts

To overcome the identified limitations, we propose two novel training methods that utilize more accurate data:

1. **Full-width TD-afterstate (TD-afterstate-full):** This method reduces variance in the training data by considering all possible future states from a given afterstate. By evaluating every possible next state, the agent can obtain a more

accurate estimate of the value of the current afterstate, leading to more stable learning.

2. **TD-afterstate with 2-ply search (2-ply TD-afterstate):** This method enhances foresight in decision-making by incorporating deeper searches into the training process. By extending the search depth to two layers (plies), the agent can account for a broader range of future scenarios, improving its ability to plan ahead and make more informed decisions.

These methods aim to improve the training of 2048 agents by generating and training with more precise and informative data, addressing the high variance and shallow search issues present in traditional TD learning approaches. The subsequent sections will delve into the detailed methodologies and theoretical foundations of these proposed methods.

3.2.2 Algorithm Design

3.2.2.1 Full-width TD-afterstate (TD-afterstate-full)

As discussed in Section 3.2.1.2, the traditional TD-afterstate training method suffers from high variance due to the randomness inherent in the game's tile placement. To mitigate this issue, we propose the Full-width TD-afterstate (TD-afterstate-full) method, an advanced training approach designed to enhance the accuracy of the neural network in the 2048 game.

Method Description The updating method for TDA-full is shown in the Fig. 3.7 . The action selection of TD-afterstate-full method is defined in Equation 3.1. After selecting an action a_t, we obtain the corresponding afterstate S'_t. Unlike the traditional TD-afterstate method, which considers only one possible next state due to the random tile addition, the TD-afterstate-full method calculates a deep value $DV(S'_t)$ by exploring all possible next states S_{t+1} that can result from the random placement of a new tile.

$$a_t = \text{argmax}_a(R(S_t, a) + V(S'_t)) \tag{3.1}$$

The deep value $DV(S'_t)$ is computed by taking the sum over all possible next states, weighting each by its probability $P(S_{t+1}|S'_t)$, and multiplying by the maximum expected

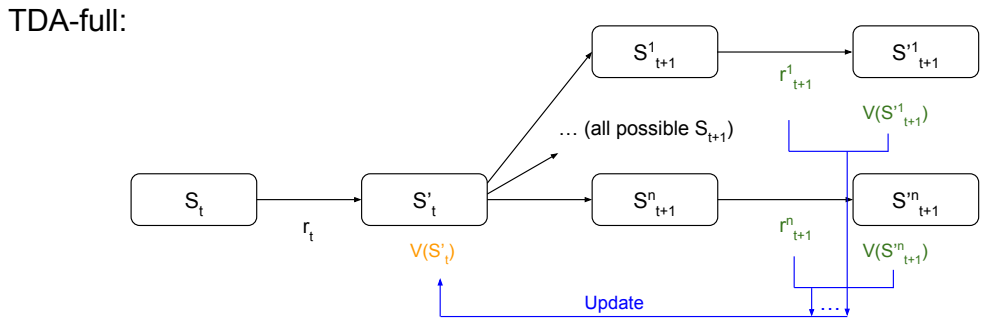


Figure 3.7: The updating method of TDA-full

value obtainable from each state. Specifically, for each possible next state S_{t+1}, we consider the highest possible sum of the immediate reward $R(S_{t+1}, a_{t+1})$ and the estimated value of the subsequent afterstate $V(S_{t+1}.move(a_{t+1}))$, where a_{t+1} represents the next action taken from state S_{t+1}.

The formula for calculating $DV(S'_t)$ is as follows:

$$DV(S'_t) = \sum_{S_{t+1}} P(S_{t+1}|S'_t) \times \max_{a_{t+1}} \left[R(S_{t+1}, a_{t+1}) + V(S'_{t+1})\right], \tag{3.2}$$

where $S'_{t+1} = S_{t+1}.move(a_{t+1})$ is the afterstate resulting from action a_{t+1} applied to state S_{t+1}.

After computing $DV(S'_t)$, the TD-afterstate-full method updates the value function $V(S'_t)$ directly using this deep value:

$$V(S'_t) \leftarrow DV(S'_t). \tag{3.3}$$

Advantages By considering all possible next states, the TD-afterstate-full method effectively reduces the variance introduced by the randomness of tile placements in the game. This comprehensive evaluation leads to more accurate and stable estimates of the value function $V(S'_t)$, enhancing the learning process of the neural network. The method ensures that the update to $V(S'_t)$ accounts for the expected outcomes over all possible future scenarios, rather than relying on a single, randomly determined next state.

Challenges The primary drawback of the TD-afterstate-full method is the increased computational complexity. Evaluating all possible next states and their corresponding afterstates requires significantly more computational resources and time. For each afterstate S'_t, the method must consider every empty tile position where a new tile can appear

and both possible tile values (typically 2 or 4), leading to a combinatorial increase in the number of states to evaluate.

Implementation Considerations Implementing the TD-afterstate-full method involves generating all possible next states S_{t+1} from the afterstate S'_t. For each empty position on the board, a new tile (2 or 4) can appear with certain probabilities (commonly 0.9 for tile 2 and 0.1 for tile 4). The probabilities $P(S_{t+1}|S'_t)$ are calculated based on these tile placement probabilities and the number of empty positions.

For each possible next state S_{t+1}, the method computes the maximum expected value over all possible actions a_{t+1}, including the immediate reward and the estimated value of the resulting afterstate $V(S'_{t+1})$. This process requires accessing the neural network to estimate $V(S'_{t+1})$ for each possible action, which can be computationally intensive.

Summary The TD-afterstate-full method addresses the high variance issue of the traditional TD-afterstate approach by incorporating a full evaluation of all possible next states in the update of the value function. While this leads to more accurate and stable learning, it also introduces significant computational challenges due to the increased number of states to evaluate. Balancing the improved accuracy with the computational cost is a key consideration in the practical implementation of this method.

The detailed experimental setup and the results demonstrating the effectiveness of the TD-afterstate-full method are presented in Section 4.4 and Section 5.3.

3.2.2.2 TD-Afterstate with 2-ply search (2-ply TD-afterstate)

Building upon the limitations identified in the traditional TD-afterstate method, particularly concerning shallow search depth, we introduce the TD-afterstate with 2-ply search (2-ply TD-afterstate) method. This novel approach is designed to enhance the training of 2048 game agents by incorporating deeper search strategies, thereby utilizing more comprehensive information from future game states.

Motivation As highlighted in Section 2.2.3, the conventional TD-afterstate method updates the network based on the value of the immediately following afterstate. However, previous studies [72, 74] have demonstrated that deeper search strategies often lead

to higher average scores. Shallow searches may not adequately capture the long-term consequences of actions, limiting the agent's ability to plan effectively.

Method Description The 2-ply TD-afterstate method extends the search depth to two layers (plies), allowing the network to update based on values that reflect deeper game dynamics. This approach provides the agent with a more foresighted perspective, potentially leading to more strategic decision-making.

The updating method is shown in the Fig. 3.8. And the update formula for the 2-ply TD-afterstate method is presented as follows:

$$V(S'_t) \leftarrow R(S_{t+1}, a_{t+1}) + DV(S'_{t+1}), \tag{3.4}$$

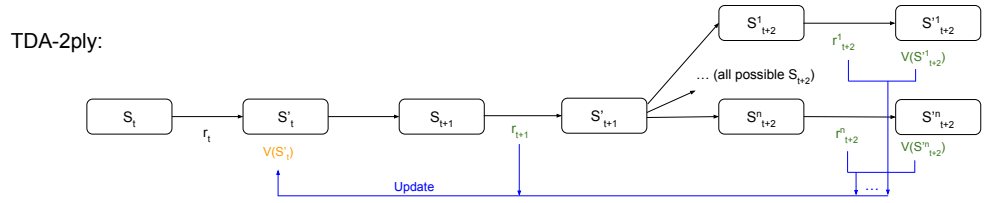


Figure 3.8: The updating method of TDA-2ply

where $DV(S'_{t+1})$ represents the deep value of the next afterstate S'_{t+1}, calculated by considering all possible subsequent states resulting from random tile placements. The deep value $DV(S'_{t+1})$ is computed similarly to Equation 3.5:

$$DV(S'_{t+1}) = \sum_{S_{t+2}} P(S_{t+2}|S'_{t+1}) \times \max_{a_{t+2}}[R(S_{t+2}, a_{t+2}) + V(S'_{t+2})], \tag{3.5}$$

where S_{t+2} represents all possible states resulting from adding a random tile to S'_{t+1}, and $S'_{t+2} = S_{t+2}.move(a_{t+2})$ is the afterstate resulting from action a_{t+2}.

Action Selection Strategies In the 2-ply TD-afterstate method, we explore two types of action selection strategies:

1. **Standard Action Selection:** Following the traditional approach as in Equation 4.2, the action a_t is chosen to maximize the immediate reward and the estimated value of the resulting afterstate $V(S'_t)$:

$$a_t = \arg\max_a [R(S_t, a) + V(S'_t)]. \tag{3.6}$$

This strategy maintains consistency with the baseline method and serves as a control for variable consistency.

2. **Deep Value-Based Action Selection:** To investigate whether actions selected through deeper searches can enhance training effectiveness, we introduce an alternative action selection strategy based on the deep value $DV(S'_t)$:

$$a_t = \arg\max_a [R(S_t, a) + DV(S'_t)]. \tag{3.7}$$

This approach selects actions that are expected to yield the highest cumulative reward over a deeper search horizon, potentially leading to more strategic gameplay.

Comparison with Baseline Method Compared to the baseline TD-afterstate method, which updates the value function using Equation 4.3 and selects actions based on immediate rewards and estimated afterstate values, the 2-ply TD-afterstate method:

- Utilizes a revised update equation (Equation 3.4) that incorporates the deep value $DV(S'_{t+1})$, accounting for a broader range of future outcomes.
- Employs two action selection strategies (Equations 4.4 and 4.5) to evaluate the impact of deeper search information on action choice.

This dual approach aims to determine whether leveraging data from more informed policies can enhance the effectiveness of training 2048 game agents.

Computational Considerations Implementing the 2-ply TD-afterstate method requires additional computational resources due to the necessity of conducting deeper searches to calculate the deep values $DV(S'_t)$ and $DV(S'_{t+1})$. The increased computational complexity arises from:

- Generating all possible next states S_{t+1} and S_{t+2} by considering every possible tile placement.
- Evaluating the maximum expected value over all possible actions a_{t+1} and a_{t+2} at each ply.
- Accessing the neural network to estimate $V(S'_{t+1})$ and $V(S'_{t+2})$ for each potential afterstate.

Despite the increased computational demands, the 2-ply TD-afterstate method is expected to provide more accurate value estimates and promote better strategic planning.

Summary The 2-ply TD-afterstate method addresses the limitations of shallow search in the traditional TD-afterstate approach by incorporating deeper search depths into the training process. By updating the value function based on the deep values of future afterstates and exploring alternative action selection strategies, this method seeks to enhance the foresight and performance of 2048 game agents.

Both the TD-afterstate-full and 2-ply TD-afterstate methods require more computational resources than the baseline TD-afterstate method. However, their underlying principles differ:

- **TD-afterstate-full** reduces variance in the training data by considering all possible next states S_{t+1} when updating $V(S'_t)$.
- **2-ply TD-afterstate** updates $V(S'_t)$ using a deeper search value $DV(S'_{t+1})$, incorporating information from an extended search horizon.

The detailed experimental setup and results demonstrating the effectiveness of the 2-ply TD-afterstate method are presented in Sections 4.4 and 5.3.

3.3 Reducing Value Gaps

The previous methods *global embedding* and *refining evaluation functions* focused on improvements from two parts: the design of neural network architectures and the training methodology. In this subsection, we further examine the limitations of existing 2048 agents and identify the presence of *value gaps* among afterstates with different ranks. The following subsections provide a detailed description of the problem and the corresponding method.

3.3.1 Overview

3.3.1.1 Areas for Improvement

In the TD-afterstate training framework, we update the current afterstate's value by using the subsequent afterstate's value combined with the reward. Consequently, once

training has converged, the 1-ply value of each afterstate should closely match its corresponding 2-ply value (which incorporates the immediate reward).

Figure 3.9 displays histograms of 1-ply and 2-ply values for afterstates from 100 games sampled from the best checkpoint of the TD-afterstate + FC-CNN model. The ranks 1 through 4 correspond to the average 1-ply values from the best to worst choices. The differences between 2-ply and 1-ply values for the 1st, 2nd, 3rd, and 4th ranks are respectively 2, 689, 7829, and 20284. Notably, as the rank increases, the gap between 2-ply and 1-ply values also grows.

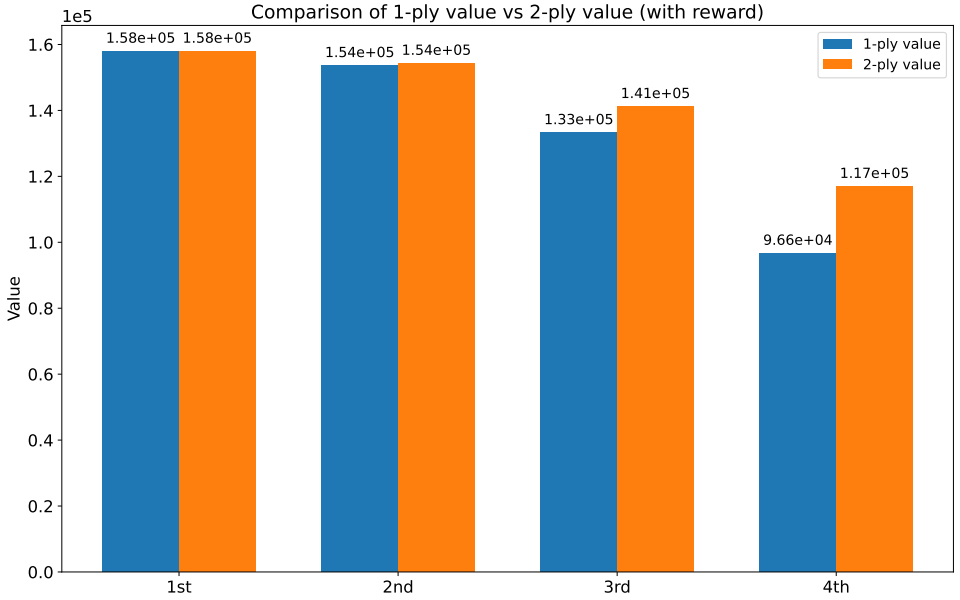
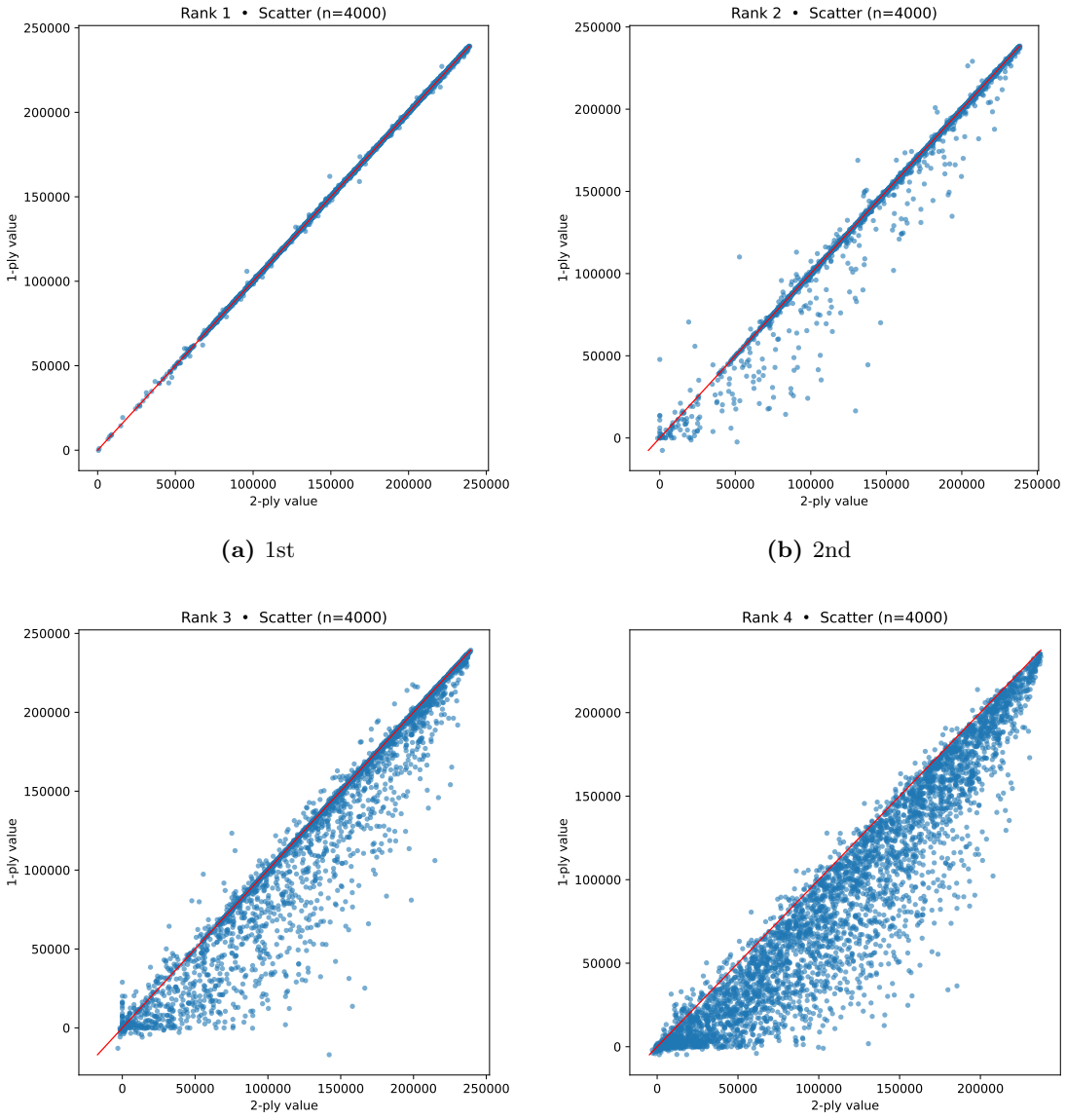


Figure 3.9: The histograms of 1-ply and 2-ply values for afterstates in different ranks

Figure 3.10 presents a scatter plot of these differences across ranks, where points near the line $y = x$ indicate afterstates whose 1-ply and 2-ply values are similar, while points located below the line (towards the lower right) denote afterstates with substantially larger 2-ply values. Here, n denotes the number of randomly sampled afterstates. Consistent with the histogram, the scatter plot further illustrates that the proportion of afterstates with 2-ply values exceeding 1-ply values increases as rank grows.

Beyond the best afterstate, all other afterstates exhibit notable mismatches between their 2-ply and 1-ply values. We hypothesize that this value gap is from the TD-afterstate training method's tendency to always select the highest-valued action, which leads to insufficient exploration of all possible afterstates. However, in the training of 2048 agents, neither ϵ-greedy nor softmax exploration strategies work for 2048 [75]. In the following parts, we will propose methods designed specifically to address this issue.



(a) 1st **(b)** 2nd

(c) 3rd **(d)** 4th

Figure 3.10: Scatter diagrams of the difference between 2-ply and 1-ply values across different ranks.

3.3.1.2 Core Concepts

To overcome the identified limitations, we designed updating methods for reducing the value gaps:

1. **All-updated TD-afterstate (TDA-all):** TDA-all is similar to the previously introduced TDA-full (in Subsection 3.2.2.1), with one critical difference: instead of updating only the best afterstate, it updates the value estimates of all possible afterstates. Our result from experiments shows that TDA-all suffers from severe overestimation problem, which causes the training failure.

2. **Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all):** To solve the overestimation problem in TDA-all, we follow the idea from the Double Deep Q-Network (DDQN) framework [76]. By integrating this principle into TDA-all, DDQN TDA-all achieves more stable learning and successfully reduces the value gaps.

3.3.2 Algorithm Design

3.3.2.1 All-updated TD-afterstate (TDA-all)

As discussed in Section 3.3.1.1, the TD-afterstate training method suffers from high value gaps between 1ply search and 2ply search value in the afterstates other than the best afterstates. To mitigate this issue, we propose the All-updated TD-afterstate (TDA-all) method.

Method Description The updating method for TDA-all is shown in the Fig. 3.11 . The action selection of TDA-all method is defined in Equation 3.8. After selecting an action a_t, we obtain the corresponding afterstate S'_t. The TDA-all method calculates a deep value $DV(S'_t)$ by exploring all possible next states S_{t+1} that can result from the random placement of a new tile.

$$a_t = \text{argmax}_a(R(S_t, a) + V(S'_t)) \tag{3.8}$$

The deep value $DV(S'_t)$ is computed by taking the sum over all possible next states, weighting each by its probability $P(S_{t+1}|S'_t)$, and multiplying by the maximum expected value obtainable from each state. Specifically, for each possible next state S_{t+1}, we consider the highest possible sum of the immediate reward $R(S_{t+1}, a_{t+1})$ and the estimated value of the subsequent afterstate $V(S_{t+1}.move(a_{t+1}))$, where a_{t+1} represents the next action taken from state S_{t+1}.

The formula for calculating $DV(S'_t)$ is as follows:

$$DV(S'_t) = \sum_{S_{t+1}} P(S_{t+1}|S'_t) \times \max_{a_{t+1}} \left[R(S_{t+1}, a_{t+1}) + V(S'_{t+1})\right], \tag{3.9}$$

where $S'_{t+1} = S_{t+1}.move(a_{t+1})$ is the afterstate resulting from action a_{t+1} applied to state S_{t+1}.

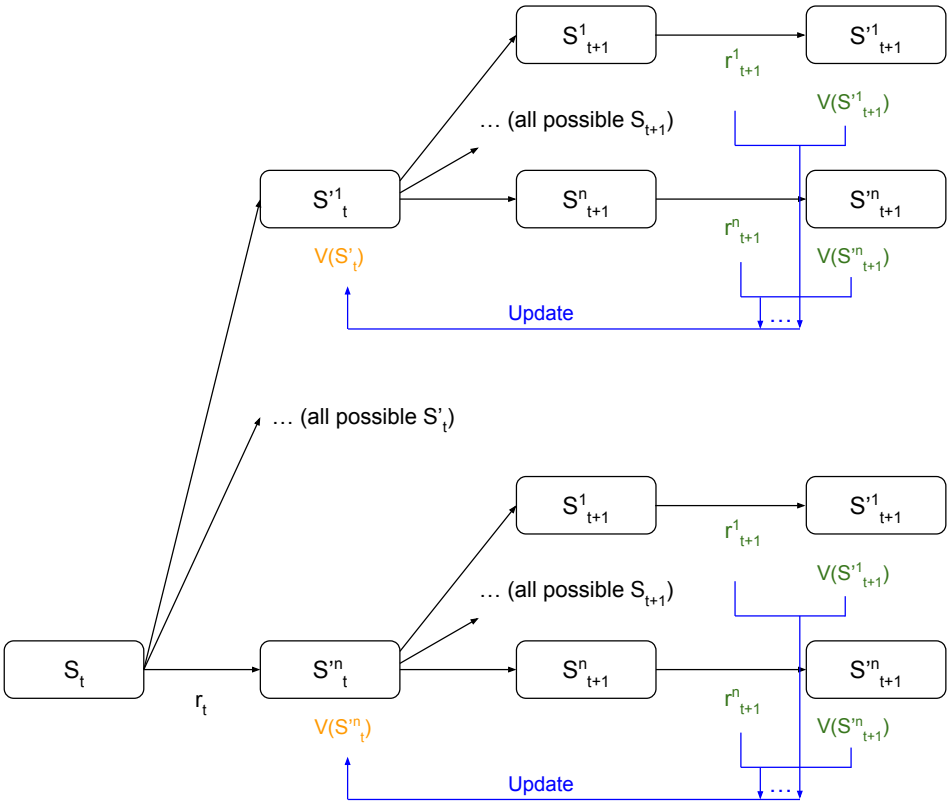


Figure 3.11: The updating method of TDA-all

After computing $DV(S'_t)$, the TDA-all method updates the value function $V(S'_t)$ for all possible afterstates S'_t directly using this deep value:

$$V(S'_t) \leftarrow DV(S'_t). \tag{3.10}$$

Compared with TDA-full method, TDA-all will update all possible afterstates instead of only update the best afterstate.

Advantages By updating all possible afterstates for each state, TDA-all tries to address the value gaps caused by the insufficient exploration problem in the standard TD-afterstate method. Although traditional exploration strategies such as ϵ-greedy and softmax have been shown to be ineffective in the context of training 2048 agents, we can still enhance value estimation by updating the values of non-best afterstates—without changing the action selection policy itself.

Challenges Updating the values of all possible afterstates will slow down the training process, as the model must perform additional forward and backward passes for each

state transition. More critically, our experiments show that TDA-all suffers from severe overestimation bias, which leads to unstable learning and eventual training failure.

3.3.2.2 Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all)

Since TDA-all suffers from severe overestimation, which leads to training failure, we adopt the idea of Double Deep Q-Network (DDQN) [76] to design the Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all). This method aims to solve the overestimation issue in TDA-all training method.

Method Description The update procedure of DDQN TDA-all is similar to that of TDA-all, but it incorporates the core idea of DDQN, namely the separation of action selection and action evaluation. In our design, we employ two neural networks, denoted as A and B. During data generation, network A is responsible for selecting actions, while network B provides the corresponding value estimates. Finally, only network A is updated, ensuring that the evaluation process remains decoupled and thereby reducing the overestimation problem.

The action selection of DDQN TDA-all is defined in Equation 3.11. After selecting an action a_t, we obtain the corresponding afterstate S'_t.

$$a_t = \text{argmax}_a \Big(R(S_t, a) + V_A(S'_t) \Big), \tag{3.11}$$

where $V_A(\cdot)$ denotes the value of the afterstate estimated by network A.

The deep value $DV(S'_t)$ in DDQN TDA-all is computed in a similar manner to TDA-all, but with the separation of action selection and action evaluation. Specifically, for each possible next state S_{t+1}, the next action a_{t+1} is selected by network A, while the evaluation of the afterstate is performed by network B. This separation follows the Double DQN principle and helps to reduce overestimation.

$$DV(S'_t) = \sum_{S_{t+1}} P(S_{t+1}|S'_t) \times \left[R(S_{t+1}, a_{t+1}) + V_B(S'_{t+1}) \right], \tag{3.12}$$

where $a_{t+1} = \arg\max_a \left(R(S_{t+1}, a) + V_A(S_{t+1}.move(a)) \right)$ is the next action chosen by network A, and $S'_{t+1} = S_{t+1}.move(a_{t+1})$ is the resulting afterstate whose value is estimated by network B.

After computing $DV(S'_t)$, the DDQN TDA-all method updates the value function $V_A(S'_t)$ of network A for all possible afterstates S'_t, while keeping network B unchanged:

$$V_A(S'_t) \leftarrow DV(S'_t). \tag{3.13}$$

Compared with TDA-all, the DDQN TDA-all method separates action selection and action evaluation by employing two networks. Only network A is updated, while network B serves as a stable evaluator to alleviate the overestimation issue observed in TDA-all.

Advantages The DDQN TDA-all method offers several advantages. First, by separating action selection and action evaluation, it effectively alleviates the overestimation problem that caused TDA-all to fail in training. Second, similar to TDA-all, it updates all possible afterstates instead of only the best one, which mitigates the insufficient exploration issue in traditional TD-afterstate and helps reduce value gaps. Finally, unlike common exploration strategies such as ϵ-greedy or softmax that are ineffective in the 2048 environment, DDQN TDA-all achieves implicit exploration by updating every afterstate without changing the action policy.

Challenges Despite its advantages, DDQN TDA-all also introduces several challenges. First, updating all afterstates and maintaining two networks significantly increases the computational cost and slows down training. Second, since our design only updates network A, the method may be more suitable as a refining strategy after the original approach has plateaued in performance, rather than as a stable training algorithm from scratch.

Chapter 4

Experiment Setup and Design

This chapter outlines the experimental setup and design employed to evaluate the proposed methods for enhancing 2048 game agents. It encompasses the hardware and software configurations, detailed descriptions of each experiment conducted, the evaluation metrics used, and considerations to ensure the reproducibility and reliability of the experiments.

4.1 Experimental Setup

This section details the hardware and software configurations utilized in our experiments, providing the necessary context for understanding the implementation and performance of the proposed methods.

4.1.1 Programming Environment and Libraries

The implementation of our models and experiments was carried out using Python 3.11, leveraging the PyTorch deep learning framework for neural network development and training. For the neural network experiments, we utilized PyTorch version 2.1.1, which provides robust support for CUDA-enabled GPU acceleration, essential for handling the computational demands of training deep neural networks.

Key libraries and tools used in our experiments include:

- **Python**: 3.11.5
- **PyTorch**: 2.1.1
- **TorchVision**: 0.16.1 (compatible with PyTorch 2.1.1)

- **NumPy**: 1.26.2
- **ONNX Runtime GPU**: 1.18.0
- **SymPy**: 1.12.1
- **Git**: 2.40.1
- **GCC (GNU Compiler Collection)**: 11.4.0
- **GNU Make**: 4.3

Additional libraries such as `requests`, `certifi`, `urllib3`, and `pillow` were used for various utility functions. The complete list of packages and their versions is managed using `conda` and is available in the environment configuration files provided in our repository.

For neural network experiments, Python and PyTorch were the primary tools due to their flexibility and extensive support for deep learning applications. PyTorch's CUDA integration facilitated efficient training on GPU hardware.

For the N-Tuple network experiments, we used C++ for its performance efficiency and control over system resources. The C++ code was compiled using GCC version 11.4.0, and the build process was managed with GNU Make version 4.3.

Version control was managed using Git version 2.40.1, ensuring consistent code management and collaboration. All experiments were conducted in a Linux environment, specifically Ubuntu 22.04.3 LTS, to maintain consistency across different machines and to leverage the robustness of Linux for computational tasks.

4.1.2 Hardware Configuration

The experiments were conducted on machines equipped with the following hardware specifications:

- **CPU**: Intel Core i7-9800X (8 cores / 16 threads) at 3.8 GHz
- **Memory**: 128 GB DDR4 RAM
- **GPU**: Two NVIDIA GeForce RTX 2080 Ti GPUs, each with 11 GB of VRAM
- **Operating System**: Ubuntu 22.04.3 LTS

Although each machine was equipped with two GPUs, our experiments utilized only a single GPU per run to ensure consistency and reproducibility. The GPUs provided the necessary computational power to train the deep neural networks efficiently, especially for models requiring extensive computation like the TD-afterstate-full and 2-ply TD-afterstate methods.

The CUDA toolkit version used was 12.3, compatible with our hardware and software configurations. CUDA facilitated GPU acceleration for neural network training, significantly reducing training times compared to CPU-only computations.

For the neural network experiments:

- GPU acceleration was enabled via PyTorch.
- Training and inference processes leveraged the high parallelism of GPUs.
- Memory management was carefully handled to accommodate the models within the 11 GB VRAM limit of the RTX 2080 Ti.

For the N-Tuple network experiments:

- Implemented in C++ to maximize execution speed and efficiency.
- GPU acceleration was not utilized, as the computational demands were sufficiently met by the CPU.
- Multithreading was employed where appropriate to leverage the multi-core CPU architecture.

The hardware configuration was selected to balance computational efficiency with resource availability, ensuring that experiments could be completed in a reasonable timeframe without compromising the integrity of the results.

4.1.3 Common Experimental Settings

All neural network-based 2048 agents were trained using the same common experimental settings to ensure a fair comparison between different models. The training framework utilized a generator-trainer model based on prior work [72], implemented using the PyTorch deep learning library. The common settings are as follows:

- **Multiprocessing:** Training data generation was handled by four parallel processes, each running gameplay using the latest neural network model. This approach accelerates data collection and improves the diversity of training samples.
- **Jump Start:** Initial game boards were allowed to contain tiles with values exceeding the standard starting tiles. This technique helps the agent experience a wider range of game states during training.
- **Restart:** After the end of each game, gameplay was restarted from the midpoint of the episode. This practice increases the number of training samples from different game stages without requiring complete game simulations each time.
- **Batch Size:** The batch size for neural network training was set to 1024, balancing memory efficiency and convergence speed.
- **Optimizer:** The Adam optimization algorithm [77] was employed for training, with an initial learning rate of 0.001.

These common settings were chosen to enhance training efficiency and model performance, following best practices established in previous studies.

4.2 Evaluation Methods

This section describes the methods used to evaluate the performance of each 2048 game agent. The evaluation employs two primary search strategies—greedy search and expectimax search—to assess the agents' decision-making capabilities and overall game performance. Additionally, the section outlines the evaluation metrics used to quantify performance and the statistical analysis methods employed to ensure the reliability of the results.

4.2.1 Evaluation Strategies

To evaluate the performance of the 2048 game agents, we employ two distinct search strategies: greedy search and expectimax search. These strategies provide insights into both immediate and long-term decision-making capabilities of the agents.

4.2.1.1 Greedy Search

Greedy search selects actions based solely on the highest predicted value from the neural network without any lookahead. The agent evaluates the immediate reward and the value of the resulting afterstate, choosing the action that maximizes the sum of these two quantities. Mathematically, the action selection at time step t is defined as:

$$a_t = \arg\max_a \left(R(S_t, a) + V(S'_t) \right), \tag{4.1}$$

where:

- S_t is the current state,
- a is a possible action,
- $R(S_t, a)$ is the immediate reward obtained by applying action a in state S_t,
- $V(S'_t)$ is the estimated value of the afterstate S'_t resulting from action a.

This strategy focuses on maximizing immediate gains without considering future consequences, providing a baseline measure of the agent's performance.

4.2.1.2 Expectimax Search

Expectimax search incorporates lookahead by evaluating potential future states up to a specified depth (ply). In our experiments, we utilize a 3-ply expectimax search. This method considers not only the immediate action but also the expected outcomes of subsequent actions, providing a more informed decision-making process that accounts for long-term rewards. The expectimax algorithm recursively applies the action selection and value estimation steps, as outlined in Algorithm 1.

Algorithm 1: Evaluate Node at Specified Ply_number

```
Function Get_Value(node, ply_number):
    if ply_number == 0 :
        return Network.predict(node)
    else if node.is_end_state() :
        return 0
    else if node.is_state() :
        return Max (Get_value(node.children[i], ply_number) + reward[i])
    else if node.is_afterstate() :
        return Average (Get_value(node.children[i], ply_number − 1) × P[i])
```

4.2.2 Evaluation Metrics

To objectively assess the performance of the agents, we focus on the following key metrics:

- **Average Game Score**: This metric represents the mean score achieved by the agent over a substantial number of game runs (e.g., 300 games) for each trained model. Given that each model is trained with a different random seed, we obtain four distinct average scores corresponding to the four seeds. The average game score serves as a reliable measure of the agent's overall performance and ability to maximize points within the game.

- 95% **Confidence Interval**: This metric provides a statistical range within which the true mean performance of the agent is expected to fall with 95% probability. By computing the interval around the sample mean score (based on repeated runs with different random seeds), we can assess the reliability and robustness of the agent's performance estimates. A narrower interval indicates more stable outcomes across runs, while a wider interval reflects higher variability.

- **95% Confidence Interval and Standard Deviation across Seeds**: This metric provides a statistical range (95% CI) for the mean performance of the agent and reports the standard deviation across different random seeds. Together, they offer insights into both the statistical reliability and robustness of the agent's performance.

These metrics provide a comprehensive understanding of the agents' performance by highlighting both their ability to achieve high scores and the consistency of these achievements across different training initializations. Focusing on the average game score and its variance allows for a clear comparison between the baseline model and the proposed global embedding methods, ensuring that any observed improvements are both significant and reliable.

4.2.3 Evaluation Procedure

To comprehensively evaluate the performance of the trained models, we implemented a structured evaluation procedure that systematically assesses each checkpoint of the models. The evaluation process is detailed as follows:

1. **Checkpoint Selection**: - For each trained model, checkpoints were selected at every 10^8 actions during the training process. Given that each model was trained for 2×10^9 actions, this resulted in 20 checkpoints per model.

2. **Simulation Runs per Checkpoint**: - For each selected checkpoint, the model was evaluated using two search strategies: 1-ply greedy search and 3-ply expectimax search. - Specifically, for each checkpoint:

   - **1-ply Greedy Search**: Conducted 75 independent game simulations. In each simulation, the agent selects actions based solely on the highest predicted value from the neural network without any lookahead.
   - **3-ply Expectimax Search**: Conducted 50 independent game simulations. In each simulation, the agent uses a 3-ply expectimax search strategy, which considers potential future states up to three steps ahead to make more informed decisions.

   - This results in a total of 125 simulations per checkpoint (75 greedy + 50 expectimax).

3. **Total Simulations**: - With 20 checkpoints per model and 4 models (each trained with a different random seed), the total number of simulations conducted is:

   $$20 \text{ checkpoints/model} \times 4 \text{ models} \times 125 \text{ simulations/checkpoint} = 10,000 \text{ game simulations}$$

4. **Performance Metrics Recording**: - For each simulation run, the following metrics were recorded:

   - **Average Greedy Game Score**: The mean score achieved over the 75 greedy(1-ply) search runs.
   - **Average Expectimax Game Score**: The mean score achieved over the 50 expectimax 3-ply search runs.
   - **95% Confidence Interval and Standard Deviation across Seeds**: The 95% confidence interval of the average scores, together with the standard deviation across different random seeds, providing insight into both the statistical reliability and the robustness of the agent's performance.

Figure 4.1 depicts a comparison between the greedy and expectimax search strategies. In each strategy, aside from the action reward, the chosen action "a" is determined solely based on the network value of the specific "Afterstates", as indicated by the blue lines in the figure.

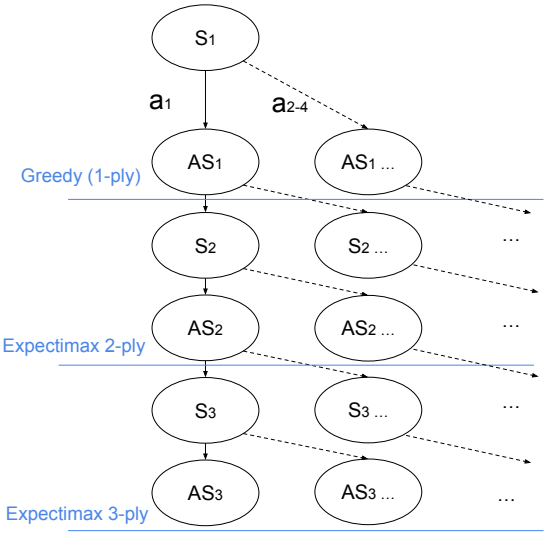


Figure 4.1: Comparison of Greedy and Expectimax Search Strategies. The maximal search depths are highlighted with blue lines, illustrating how actions are selected based on the network's valuation of afterstates.

4.3 Experimental Setup for Global Embedding Methods

In this section, we present the experimental evaluation of the proposed global embedding methods aimed at enhancing the performance of the 2048 game agent. The global

embedding methods implemented include BP-CNN, FC-CNN, CNN-CNN, and T-CNN, each introducing distinct mechanisms to integrate global contextual information into the baseline CNN22 model. The experimental setup for these methods is detailed below.

4.3.1 Overview

The primary goal of the global embedding methods is to enhance the neural network's ability to comprehend and utilize global contextual information alongside local feature extraction. Each method achieves this through different architectural modifications:

- **BP-CNN (Bypass CNN)**: Augments the input representation by adding positional information to the tile value encoding.
- **FC-CNN (Fully Connected Embedding CNN)**: Introduces fully connected layers to generate a global embedding vector, which is merged with the input before convolutional layers.
- **CNN-CNN (Convolutional Embedding CNN)**: Adds additional convolutional layers dedicated to extracting global contextual features.
- **T-CNN (Transformer Embedding CNN)**: Integrates a Transformer encoder layer to capture complex global dependencies.

Each method was evaluated under the same common experimental settings to ensure a fair comparison with the baseline CNN22 model. During gameplay, the agent selects actions using a greedy strategy based on the estimated values from the network. The action at time step t is chosen as:

$$a_t = \arg\max_a \left(R(S_t, a) + V(S'_t) \right), \tag{4.2}$$

S_t is the current state, a is a possible action, $R(S_t, a)$ is the immediate reward obtained by applying action a in state S_t, and $V(S'_t)$ is the estimated value of the afterstate S'_t resulting from action a.

The network is trained using the TD-afterstate learning method, where the value function is updated based on the temporal difference between successive afterstates. The update rule is:

$$V(S'_t) \leftarrow R(S_{t+1}, a_{t+1}) + V(S'_{t+1}), \tag{4.3}$$

$R(S_{t+1}, a_{t+1})$ is the reward obtained by applying action a_{t+1} in state S_{t+1}, and $V(S'_{t+1})$ is the estimated value of the next afterstate.

4.3.2 Global Embedding Methods Implementation

In this subsection, we will show our experiment designing details for each method. Each method will be trained with 4 different random seed. We will first train each method for 5×10^8 actions and show the greedy average score for them. Then, we will train the best method for 2×10^9 actions to futher show the performance.

4.3.2.1 BP-CNN (Bypass CNN)

Implementation Details BP-CNN augments the input tensor by concatenating additional channels representing the x and y coordinates of each tile. Specifically, the input tensor size is increased from $4 \times 4 \times 18$ to $4 \times 4 \times 26$. This augmentation provides the network with explicit spatial context, enhancing its ability to recognize and utilize spatial patterns critical for strategic decision-making.

Training Procedure The only modification was the augmented input tensor, ensuring that the network could leverage positional information effectively.

4.3.2.2 FC-CNN (Fully Connected Embedding CNN)

Implementation Details FC-CNN introduces two fully connected layers to generate a global embedding vector from the flattened input tensor. This global embedding is then reshaped and concatenated with the original input before being fed into the convolutional layers. This approach allows the network to capture and integrate comprehensive game state information more effectively.

Training Procedure The addition of fully connected layers provided a robust mechanism for embedding global context, enhancing the network's decision-making capabilities.

4.3.2.3 CNN-CNN (Convolutional Embedding CNN)

Implementation Details CNN-CNN adds two additional convolutional layers dedicated to extracting global contextual features. These layers process the augmented input tensor and produce a global embedding that is incorporated into the main convolutional

layers. This design enables the network to capture complex spatial dependencies and global patterns across the game board.

Training Procedure The additional convolutional layers facilitated enhanced global feature extraction, improving the agent's strategic performance.

4.3.2.4 T-CNN (Transformer Embedding CNN)

Implementation Details T-CNN integrates a Transformer encoder layer to capture global dependencies. The Transformer processes the enriched input sequence, leveraging its self-attention mechanism to model complex relationships across the entire board. The output embedding from the Transformer is then combined with the original input for subsequent processing.

Training Procedure The inclusion of a Transformer layer enabled the network to effectively capture and utilize global contextual information, enhancing strategic decision-making.

4.4 Experimental Setup for Refining Evaluation Functions

4.4.1 Overview

This section outlines the experimental design and procedures used to evaluate the enhanced training methods introduced in Section 3.2. We focus on assessing the performance improvements of the 2-ply TD-afterstate method with various action selection strategies and the TD-afterstate-full method. The valuation network utilized in these experiments is the FC-CNN, identified as the best-performing network in the previous experimental setup. Two training approaches were employed: training from scratch and fine-tuning from pretrained checkpoints. This comprehensive evaluation ensures a thorough understanding of the effectiveness and robustness of the proposed methods.

4.4.2 Training Approaches

To evaluate the effectiveness of the proposed enhanced methods, we adopted the following two training approaches:

1. **Training from Scratch**: Models were initialized with random weights and trained using the enhanced methods without any prior knowledge. This approach allows

us to observe the initial learning capabilities and convergence behavior of the methods.

2. **Fine-Tuning from Pretrained Checkpoints**: Models were initialized using the parameters from the final checkpoint of the baseline FC-CNN + TD-afterstate model trained over 2×10^9 actions. The models were then further trained using the enhanced methods. This approach leverages previously learned representations to accelerate training and improve performance while conserving computational resources.

4.4.2.1 Purpose of Each Approach

Training from scratch was primarily conducted to demonstrate the initial performance of the various enhanced methods, providing a baseline for how well the methods can learn without any prior knowledge. Fine-tuning, on the other hand, represents the final performance of these methods when leveraging the knowledge acquired from the baseline model. This approach is crucial for achieving higher performance levels while conserving computational resources. To obtain more accurate results, a larger number of episodes were tested during the fine-tuning phase when evaluating the average scores of the experimental outcomes.

4.4.3 Experimental Procedures

4.4.3.1 2-Ply TD-Afterstate Method Experiments

Initially, we focused on the 2-ply TD-afterstate method, experimenting with different action selection strategies to determine the most effective approach. The following steps were undertaken:

Training Process

- **Training Duration**: Each model was trained for 5×10^8 actions.
- **Training Approaches**: Both training from scratch and fine-tuning were applied to each variant of the 2-ply TD-afterstate method.
- **Checkpoint Intervals**: Checkpoints were saved every 5×10^7 actions to monitor training progress.

Evaluation Procedure At each checkpoint, the following evaluations were conducted:

- **Training from Scratch**:
  - **1-Ply Greedy Search**: Conducted 75 independent game simulations per seed using 1-ply greedy search, totaling 300 simulations across the four random seeds.
  - **3-Ply Expectimax Search**: Conducted 50 independent game simulations per seed using 3-ply expectimax search, totaling 200 simulations across the four random seeds.
- **Fine-Tuning**:
  - **1-Ply Greedy Search**: Conducted 75 independent game simulations per seed, totaling 300 simulations.
  - **3-Ply Expectimax Search**: Conducted 50 independent game simulations per seed, totaling 200 simulations.

These evaluations allowed us to compare the performance of different action selection strategies within the 2-ply TD-afterstate method and to assess the benefits of fine-tuning over training from scratch.

4.4.3.2 TD-Afterstate-Full Method Experiments

Following the 2-ply experiments, we extended our investigation to include the TD-afterstate-full method. The experimental setup was similar to that of the 2-ply TD-afterstate method.

Training Process

- **Training Duration**: Models were trained for 5×10^8 actions.
- **Training Approaches**: Both training from scratch and fine-tuning were applied.
- **Checkpoint Intervals**: Checkpoints were saved every 5×10^7 actions.

Evaluation Procedure At each checkpoint, the following evaluations were conducted:

- **1-Ply Greedy Search**: Conducted 75 simulations per seed using 1-ply greedy search, totaling 300 simulations.
- **3-Ply Expectimax Search**: Conducted 50 simulations per seed using 3-ply expectimax search, totaling 200 simulations.

4.4.4 Action Selection Strategies in 2-Ply TD-Afterstate Method

In the 2-ply TD-afterstate method, we explore two types of action selection strategies:

1. **Standard Action Selection:** Following the traditional approach as described in Equation 4.4, the action a_t is chosen to maximize the immediate reward and the estimated value of the resulting afterstate $V(S'_t)$:

$$a_t = \arg\max_a [R(S_t, a) + V(S'_t)]. \tag{4.4}$$

   This strategy maintains consistency with the baseline method and serves as a control to ensure variable consistency across experiments.

2. **Deep Value-Based Action Selection:** To investigate whether actions selected through deeper searches can enhance training effectiveness, we introduce an alternative action selection strategy based on the deep value $DV(S'_t)$:

$$a_t = \arg\max_a [R(S_t, a) + DV(S'_t)]. \tag{4.5}$$

   This approach selects actions that are expected to yield the highest cumulative reward over a deeper search horizon, potentially leading to more strategic gameplay.

By evaluating these strategies, we aim to determine which approach leads to better performance in terms of average game score and consistency across different random seeds. The Standard Action Selection serves as a baseline for comparison, while the Deep Value-Based Action Selection explores the potential benefits of incorporating deeper value estimations into the decision-making process.

4.5 Experimental Setup for Reducing Value Gaps

4.5.1 Overview

This section presents the experimental design and procedures used to evaluate the enhanced training methods introduced in Section 3.3. Our focus is on reducing the value gaps that arise in the TD-afterstate training method. For all experiments, we employ the FC-CNN architecture, which was identified as the best-performing network in the previous experimental setup. Both methods are trained in a refining stage, starting from the best 1-ply score checkpoint obtained with TD-afterstate.

4.5.2 Experimental Procedures

4.5.2.1 TDA-all Method Experiments

To evaluate the effectiveness of the TDA-all method, we conducted a series of experiments based on the refining setup described in the overview. The training started from the best 1-ply checkpoint obtained with TD-afterstate, and the network was further optimized using the TDA-all updating method. During this stage, all possible afterstates were updated at each step, allowing us to directly assess whether expanding updates beyond the best afterstate can reduce the observed value gaps.

Training Process

- **Initialization**: The TDA-all agent was initialized from the checkpoint that achieved the highest 1-ply score in the TD-afterstate training. This provides a strong baseline for refining instead of starting from random initialization.
- **Training Length**: We trained the agent with four different random seeds. Each run was trained for 3×10^8 actions. The relatively short training horizon was chosen because preliminary experiments showed severe performance degradation, so further training was not pursued.
- **Checkpoint Recording**: Each checkpoint was recorded every 0.5×10^8 actions to monitor the training dynamics.

Evaluating Process

- **1-Ply Greedy Search**: Conducted 75 independent game simulations per seed (300 in total). This provides a stable estimate of the agent's performance at different training stages.
- **1-Ply Value Check**: Conducted 75 independent game simulations per seed (300 in total). For each game, we averaged the estimated values of all chosen afterstates. This evaluation was only applied to the final checkpoint.

4.5.2.2 DDQN TDA-all Method Experiments

To evaluate the effectiveness of the DDQN TDA-all method, we conducted experiments under the refining setup described in the overview. Similar to the TDA-all experiments, the training started from the checkpoint that achieved the highest 1-ply score in the TD-afterstate training. By incorporating the DDQN updating strategy, we aimed to test whether the separation of action selection and action evaluation could mitigate the overestimation issue observed in TDA-all and further reduce value gaps.

Training Process

- **Initialization**: The DDQN TDA-all agent was initialized from the checkpoint that achieved the highest 1-ply score in the TD-afterstate training. This ensures a strong baseline for refining rather than starting from random initialization.
- **Training Length**: We trained the agent with four different random seeds. Each run was trained until the performance reached a plateau, with a total of 2.5×10^9 actions.
- **Checkpoint Recording**: Each checkpoint was recorded every 5×10^7 actions to monitor the training dynamics.

Evaluating Process

- **1-Ply Greedy Search**: Conducted 75 independent game simulations per seed (300 in total). This provides a stable estimate of the agent's performance.
- **Value Gap Analysis**: At each checkpoint, we performed 100 independent game simulations per seed (400 in total). For each game, we recorded the estimated values of both 1-ply and 2-ply afterstates and computed their differences. This analysis was designed to quantify the reduction of value gaps during training.

4.5.2.3 DDQN TDA-all with TDA-full Experiments

To further investigate the effect of combining methods, we selected the checkpoint with the best 1-ply score from DDQN TDA-all training and continued training it with the TDA-full method. This setup allows us to evaluate whether refining with TDA-full can further improve performance while observing the behavior of value estimation.

Training Process

- **Initialization**: The agent was initialized from the best 1-ply checkpoint obtained during DDQN TDA-all training. This ensures that the refinement starts from a stable and non-collapsed model.
- **Training Length**: We trained the agent with four different random seeds for a total of 5×10^8 actions under the TDA-full method.
- **Checkpoint Recording**: Each checkpoint was recorded every 5×10^7 actions to track the progress of the refinement process.

Evaluating Process

- **1-Ply Greedy Search**: Conducted 75 independent game simulations per seed (300 in total) to provide a stable estimate of the baseline greedy performance.
- **3-Ply Expectimax Search**: Conducted 50 independent game simulations per seed (200 in total) to measure deeper lookahead performance.
- **Value Gap Analysis**: At each checkpoint, 100 independent game simulations per seed (400 in total) were run to record both 1-ply and 2-ply afterstate values. The differences were calculated to assess whether refining with TDA-full reintroduced value gaps.

Chapter 5

Results and Discussion

5.1 Overview

In this chapter, we present and analyze the experimental results obtained from our proposed methods. The chapter is divided into two main sections. The first section focuses on the Global Embedding methods, where we evaluate and compare the performance of different neural network architectures, including CNN22, BP-CNN, CNN-CNN, FC-CNN, and T-CNN. The second section discusses the results of the Generating and Training with Accurate Data approach. We provide detailed analyses of the models' performance, including average scores, standard deviations, and the probability of achieving specific tile values during gameplay.

5.2 Experimental Results of Global Embedding Methods

5.2.1 Performance Metrics Overview

We use the average score to evaluate the performance of the different models. The mean score indicates the average performance of the model, while the confidence interval reflects the consistency of the results.

5.2.2 Performance under Greedy Search

In this subsection, we will first show the performance for each method under greedy search. Figure 5.1 shows the greedy average score with 95% confidence interval. From the figure, we can find that FC-CNN method can get the best average score in greedy search.

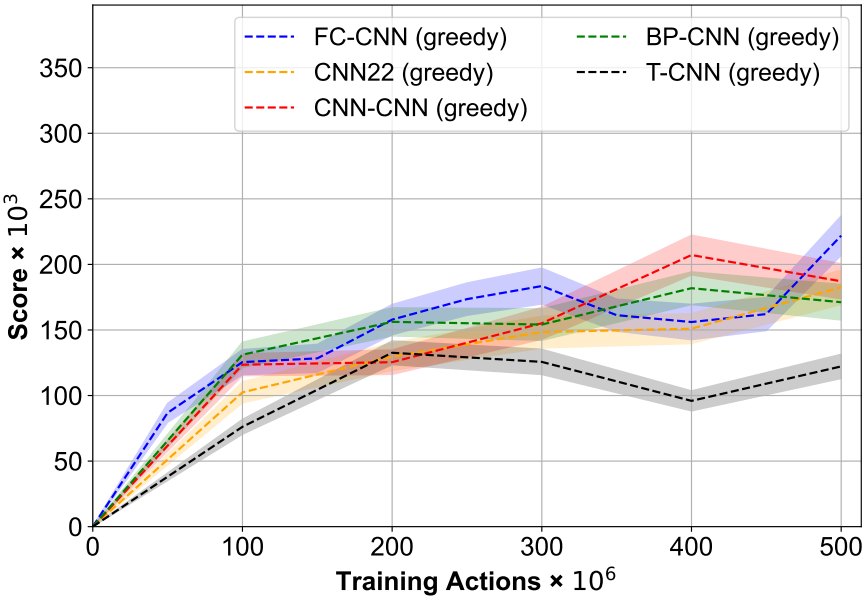


Figure 5.1: Score analysis of the each model using greedy (1-ply) search strategies.

5.2.3 Performance of FC-CNN

In this subsection, we will show the performance of our best method FC-CNN in longer training. Figure 5.2 shows the average score with 95% confidence interval in both greedy 1-ply search and expectimax 3-ply search.

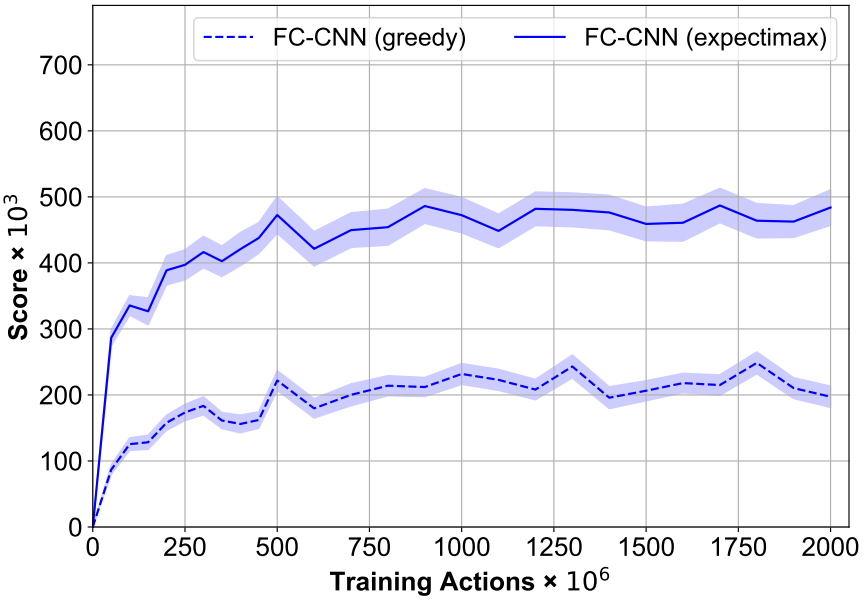


Figure 5.2: Score analysis of the FC-CNN model using 1-ply and 3-ply search strategies.

Table 5.1 and Table 5.2 show the detailed performance for each method. FC-CNN

can get the best greedy score 248, 575 and the best expectimax 3-ply score 486, 999. These result are much better than our baseline method CNN22 with greedy score 248, 575 and expectimax 3-ply score 486, 999 shown in [72].

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	86823.07±18572.84	90.3%	83.0%	69.3%	40.0%	4.3%	0.0%	0.0%
100	125491.49±9009.83	94.7%	90.3%	82.0%	60.0%	14.0%	0.0%	0.0%
150	128345.32±28493.11	96.7%	93.7%	81.0%	53.3%	17.7%	0.3%	0.0%
200	157907.61±28514.71	96.0%	93.0%	87.3%	72.0%	23.3%	0.7%	0.0%
250	173514.35±26340.22	99.0%	96.0%	90.7%	72.3%	31.0%	1.0%	0.0%
300	183497.93±32506.76	97.0%	93.3%	87.7%	71.7%	37.3%	1.7%	0.0%
350	161285.40±26119.77	98.0%	95.7%	89.3%	70.7%	25.0%	1.0%	0.0%
400	156054.71±65530.75	95.3%	88.0%	79.0%	61.0%	29.7%	0.3%	0.0%
450	162120.21±19199.14	96.0%	93.0%	83.7%	65.7%	30.3%	0.7%	0.0%
500	221931.61±29386.88	99.3%	98.3%	95.0%	79.3%	47.7%	4.3%	0.0%
600	179707.73±65129.62	97.3%	92.0%	86.3%	66.7%	36.7%	2.0%	0.0%
700	200172.72±39080.43	96.7%	94.0%	87.0%	73.0%	39.3%	4.0%	0.0%
800	214001.59±21045.46	99.7%	97.0%	95.0%	78.3%	45.0%	3.0%	0.0%
900	212006.35±8004.90	98.7%	95.3%	90.7%	78.3%	45.7%	2.7%	0.0%
1000	231810.56±38304.73	99.0%	96.3%	92.7%	81.7%	50.3%	4.7%	0.0%
1100	222841.72±3699.94	98.0%	96.0%	89.7%	79.3%	46.7%	4.0%	0.0%
1200	208127.67±41174.76	97.7%	95.7%	89.7%	76.3%	41.3%	4.7%	0.0%
1300	243124.49±37851.66	99.3%	98.0%	92.7%	81.3%	50.7%	8.3%	0.0%
1400	195821.96±42026.87	96.3%	94.0%	89.0%	70.7%	38.7%	4.7%	0.0%
1500	206400.49±20282.42	96.7%	94.0%	87.3%	76.0%	44.3%	3.3%	0.0%
1600	217994.05±33070.31	97.7%	95.7%	91.7%	81.7%	47.0%	3.0%	0.0%
1700	215019.60±41462.85	97.7%	93.0%	88.0%	77.0%	50.3%	3.0%	0.0%
1800	248575.16±34793.57	98.7%	98.0%	94.7%	84.3%	53.0%	7.3%	0.0%
1900	210382.09±26520.70	98.7%	95.3%	91.7%	74.7%	45.3%	4.7%	0.0%
2000	197239.25±50388.74	94.7%	88.7%	82.7%	72.0%	41.3%	4.3%	0.0%

Table 5.1: Results for model: FC-CNN (1-ply)

5.3 Experimental Results of Refining Evaluation Function

5.3.1 Performance Metrics Overview

In this section, we present the experimental results of the methods that involve generating and training with accurate data. We evaluate the models using the mean score, standard deviation, and the probabilities of achieving specific tile values (from 1024 to 65536). The experiments include both training from scratch and fine-tuning approaches. The models are evaluated using 1-ply and 3-ply search strategies.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	286617.14±10275.26	100.0%	100.0%	99.5%	97.0%	74.5%	0.0%	0.0%
100	335564.30±19813.77	100.0%	100.0%	99.5%	97.5%	83.5%	5.5%	0.0%
150	326683.70±48264.32	100.0%	99.0%	98.5%	92.0%	77.5%	11.5%	0.0%
200	388799.30±14892.57	100.0%	100.0%	99.0%	97.0%	87.0%	22.5%	0.0%
250	397104.84±32657.22	100.0%	100.0%	100.0%	99.5%	86.0%	23.0%	0.0%
300	416435.86±60873.82	100.0%	100.0%	100.0%	99.0%	88.0%	28.5%	0.0%
350	402538.08±15070.12	100.0%	100.0%	99.0%	98.0%	86.0%	25.5%	0.0%
400	420774.16±53498.30	100.0%	100.0%	99.5%	97.5%	87.0%	30.5%	0.0%
450	437769.12±15237.99	100.0%	100.0%	99.5%	98.5%	94.5%	32.5%	0.0%
500	472487.68±42058.25	100.0%	100.0%	100.0%	99.0%	89.0%	42.0%	0.0%
600	421362.04±74614.90	99.5%	99.0%	99.0%	97.0%	86.0%	29.5%	0.0%
700	449641.96±37013.16	100.0%	100.0%	99.0%	96.0%	90.0%	38.0%	0.0%
800	454086.24±48623.65	100.0%	100.0%	99.0%	95.5%	90.5%	37.0%	0.0%
900	486148.04±31601.77	100.0%	100.0%	100.0%	100.0%	92.0%	44.5%	0.0%
1000	472282.20±27315.72	100.0%	100.0%	100.0%	97.5%	91.5%	41.5%	0.0%
1100	448419.64±29752.35	100.0%	100.0%	99.5%	98.5%	91.0%	33.5%	0.0%
1200	481938.56±43648.75	100.0%	100.0%	99.5%	99.5%	95.0%	38.5%	0.0%
1300	480347.90±43845.95	100.0%	99.5%	99.5%	99.5%	94.0%	44.0%	0.0%
1400	476390.62±30083.96	100.0%	100.0%	100.0%	98.0%	92.0%	43.0%	0.0%
1500	459019.00±32298.37	100.0%	100.0%	100.0%	98.5%	92.0%	39.0%	0.0%
1600	460838.56±20029.32	100.0%	100.0%	99.0%	97.5%	87.0%	42.0%	0.0%
1700	486998.62±32749.12	100.0%	100.0%	100.0%	99.0%	92.5%	45.0%	0.0%
1800	463901.06±29105.80	100.0%	100.0%	100.0%	99.5%	90.0%	39.5%	0.0%
1900	462563.08±29867.42	100.0%	99.5%	99.0%	98.5%	94.5%	40.0%	0.0%
2000	483897.14±52442.22	100.0%	100.0%	100.0%	97.5%	90.5%	46.5%	0.0%

Table 5.2: Results for model: FC-CNN (3-ply)

5.3.2 TDA-2ply Method Results

5.3.2.1 Training from Scratch

The TDA-2ply method is trained from scratch using the FC-CNN architecture. Figure 5.3 shows the score analysis of the model.

Table 5.3 and Table 5.4 presents the detailed results, including the mean score, standard deviation, and the probabilities of achieving specific tile values.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	100462.69±28126.65	97.7%	92.0%	76.3%	43.3%	9.0%	0.0%	0.0%
100	106668.52±14907.26	92.3%	86.3%	74.7%	49.0%	11.3%	0.0%	0.0%
150	154903.35±17435.69	95.3%	91.3%	80.7%	59.7%	27.7%	0.7%	0.0%
200	150043.53±32935.40	97.7%	92.7%	83.3%	59.3%	25.3%	1.0%	0.0%
250	170497.37±38388.89	96.3%	91.3%	84.7%	66.7%	35.3%	1.0%	0.0%
300	187928.63±25668.91	97.3%	95.3%	89.3%	75.3%	37.3%	1.3%	0.0%

Table 5.3: Results for model: TDA-2ply (1-ply)

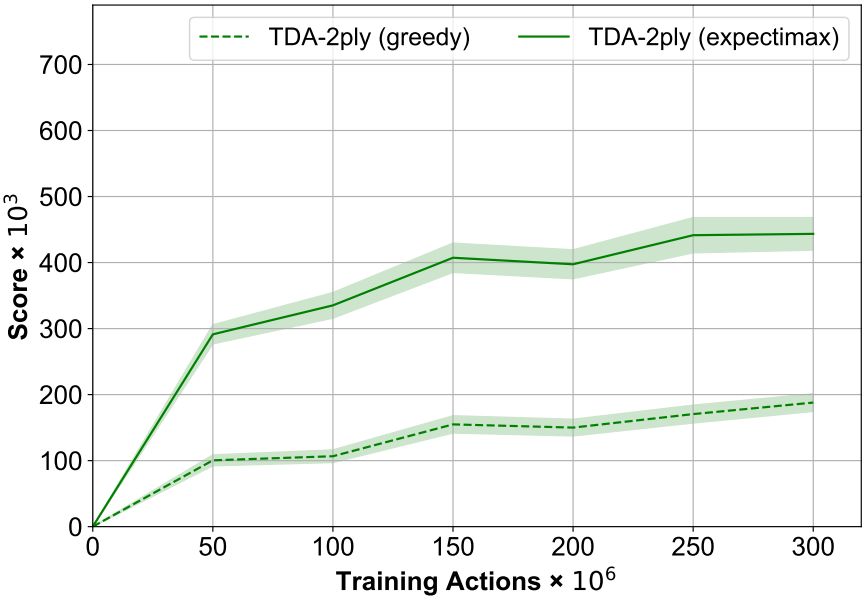


Figure 5.3: Score analysis of the 2-ply TD-afterstate model using 1-ply and 3-ply search strategies.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	291273.80±43955.81	100.0%	100.0%	99.0%	96.5%	72.0%	2.0%	0.0%
100	335165.30±22569.69	100.0%	100.0%	98.5%	94.5%	78.5%	10.5%	0.0%
150	407278.20±25737.46	100.0%	100.0%	98.5%	98.0%	89.5%	25.5%	0.0%
200	397389.88±53147.95	99.5%	99.5%	99.0%	97.5%	88.0%	23.0%	0.0%
250	441442.40±30571.28	100.0%	100.0%	99.0%	97.0%	88.5%	35.5%	0.0%
300	443455.54±9278.81	100.0%	100.0%	99.5%	98.5%	88.5%	36.0%	0.0%

Table 5.4: Results for model: TDA-2ply (3-ply)

5.3.2.2 Fine-Tuning Approach

We fine-tune the 2-ply TD-afterstate model based on a pre-trained FC-CNN model. Figure 5.4 illustrates the score analysis after fine-tuning.

Table 5.5 and Table 5.6 provides the detailed results. TDA-2ply with refining method can reach $260,117.35$ for highest 1-ply result and $512,997.56$ for highest 3-ply result

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
2050	208623.95±51020.76	96.7%	94.3%	91.7%	82.3%	43.0%	2.7%	0.0%
2100	204190.07±59732.97	97.3%	94.0%	89.0%	75.0%	42.0%	3.0%	0.0%
2150	224610.17±27005.61	98.0%	95.0%	89.0%	76.3%	47.3%	7.0%	0.0%
2200	227819.08±87315.50	98.0%	94.0%	87.7%	75.0%	50.0%	7.3%	0.0%
2250	260117.35±9108.62	98.3%	97.7%	94.7%	87.3%	58.3%	8.0%	0.0%
2300	247929.75±31863.09	96.7%	93.7%	89.7%	82.3%	54.7%	7.3%	0.0%

Table 5.5: Results for model: TDA-2ply-refining (1-ply)

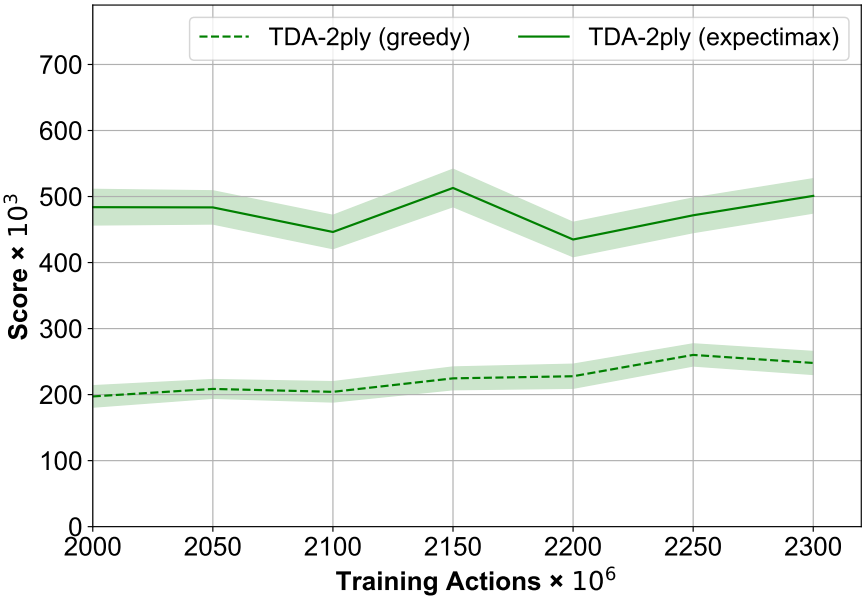


Figure 5.4: Score analysis of the 2-ply TD-afterstate model with fine-tuning using 1-ply and 3-ply search strategies.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
2050	483532.36±25167.10	100.0%	100.0%	99.5%	99.5%	94.5%	43.5%	0.0%
2100	446309.40±36122.90	100.0%	100.0%	100.0%	98.0%	90.0%	37.0%	0.0%
2150	512997.56±45863.78	100.0%	100.0%	100.0%	99.5%	92.0%	50.5%	0.0%
2200	434916.44±113815.24	100.0%	99.5%	99.5%	98.5%	95.5%	32.0%	0.0%
2250	471591.20±18535.04	100.0%	100.0%	99.5%	98.5%	91.5%	41.5%	0.0%
2300	500981.80±43006.03	100.0%	100.0%	100.0%	99.5%	95.0%	47.5%	0.0%

Table 5.6: Results for model: TDA-2ply-refining (3-ply)

Table 5.7: The network value for afterstates

	Network Value	Score (Greedy Search)
TDA-M-2ply	3.92×10^{14}	2.41×10^{3}
TDA-2ply	1.75×10^{5}	2.60×10^{5}

The fine-tuning approach significantly improves the model's performance, achieving higher mean scores and increased probabilities of reaching higher tile values.

5.3.2.3 Using moves with 2-ply search in TDA-2ply

In this part, we will show the result of TD-afterstate with moves after 2-ply search (hereinafter TDA-M-2ply). Table 5.7 shows the average network value and the average greedy search score.

From the table, we observed severe overestimation issues. With TD learning and its variants, those evaluation values should match with the average score of the greedy play.

However, the value after TDA-M-2ply was larger by several magnitude, with which we confirmed the presence of significant overestimation issues.

Overestimation issues have been extensively studied in the field of reinforcement learning as highlighted in [78] and [79]. Specifically, the issue was addressed in the context of Atari games using Q-learning, where the Double Q-learning algorithm effectively mitigated the overestimation [80, 81], as detailed by [82]. The principles of Double Q-learning might offer a potential solution to address the overestimation issues observed with the TDA-M-2ply method, we let the application Double (Q-)learning and its evaluation be our future work.

5.3.3 TD-Afterstate-Full Method Results

5.3.3.1 Training from Scratch

The TD-afterstate-full method extends the 2-ply TD-afterstate method by considering the full afterstate. Figure 5.5 presents the score analysis.

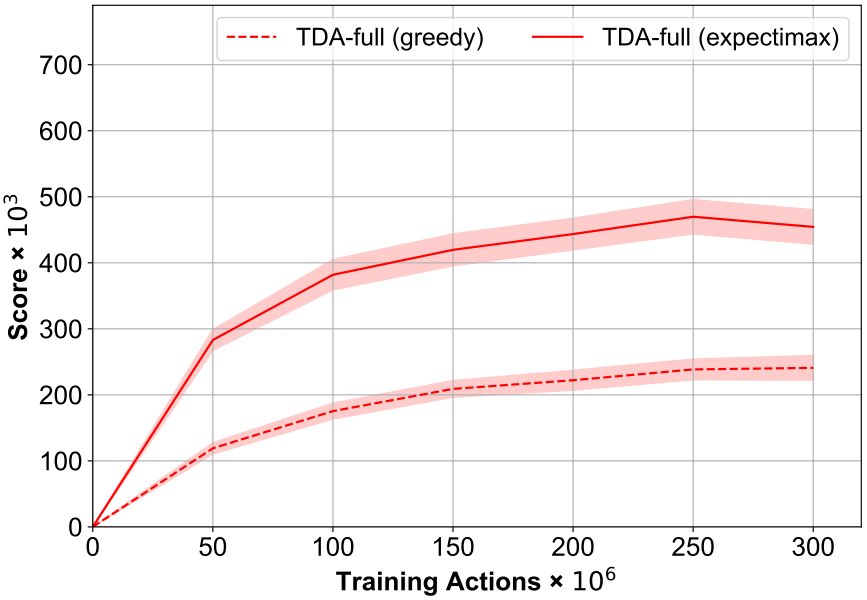


Figure 5.5: Score analysis of the TD-afterstate-full model using 1-ply and 3-ply search strategies.

Table 5.8 and Table 5.9 show the detailed results for TDA-full with training from scratch.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	119027.07±16633.57	97.7%	93.7%	83.3%	56.3%	10.3%	0.0%	0.0%
100	175367.73±38843.98	98.7%	96.7%	90.7%	74.0%	32.3%	0.3%	0.0%
150	209030.67±15512.43	99.0%	98.7%	95.0%	82.0%	45.3%	1.3%	0.0%
200	222132.87±29498.06	98.0%	96.7%	92.7%	78.3%	49.7%	4.0%	0.0%
250	238519.33±23813.62	99.3%	98.0%	94.0%	82.3%	54.0%	5.3%	0.0%
300	240896.08±21316.22	97.3%	94.7%	87.7%	78.3%	51.3%	9.0%	0.0%

Table 5.8: Results for model: TDA-full (1-ply)

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
50	283052.22±33620.02	100.0%	99.5%	98.0%	91.5%	69.0%	2.5%	0.0%
100	381856.68±46535.73	100.0%	100.0%	99.0%	96.5%	82.5%	21.5%	0.0%
150	419636.04±25865.40	100.0%	100.0%	98.5%	97.0%	90.0%	27.5%	0.0%
200	443519.80±29741.82	100.0%	100.0%	100.0%	99.0%	91.5%	33.0%	0.0%
250	469744.76±39442.11	100.0%	100.0%	100.0%	99.5%	91.0%	40.5%	0.0%
300	454379.50±43679.72	100.0%	100.0%	99.5%	98.0%	90.5%	38.0%	0.0%

Table 5.9: Results for model: TDA-full (3-ply)

5.3.3.2 Fine-Tuning Approach

Fine-tuning the TD-afterstate-full model further enhances its performance. Figure 5.6 shows the score analysis after fine-tuning.

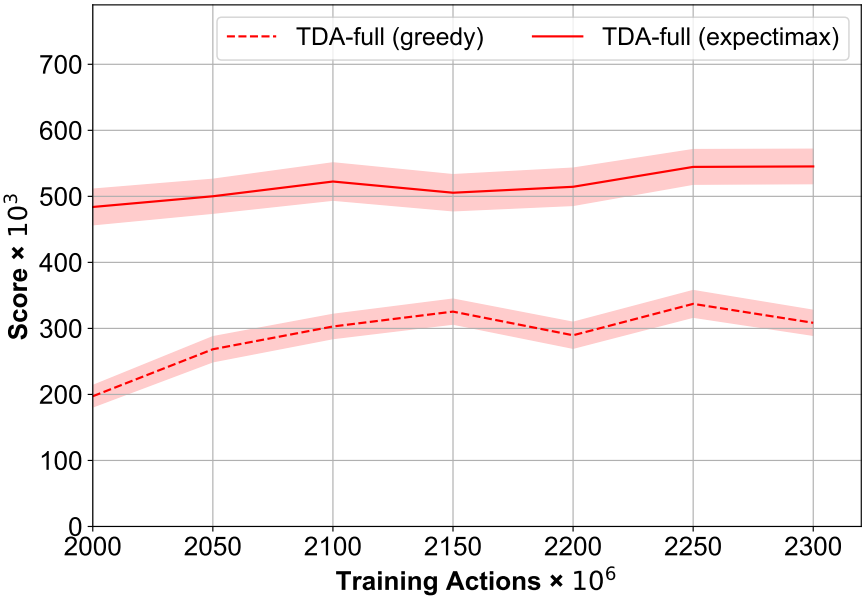


Figure 5.6: Score analysis of the TD-afterstate-full model with fine-tuning using 1-ply and 3-ply search strategies.

Table 5.10 and Table 5.11 provide the detailed results. With the refining method, TDA-full can get the highest 1-ply result at 337, 167.76 and the highest 3-ply result at 544, 596.52

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
2050	268410.49±44396.64	97.0%	94.3%	93.3%	85.7%	59.7%	12.0%	0.0%
2100	302894.45±31924.79	98.7%	97.3%	93.7%	89.7%	71.0%	13.3%	0.0%
2150	325426.63±23418.75	99.7%	98.7%	96.7%	93.0%	74.3%	16.7%	0.0%
2200	289587.16±49703.17	97.0%	96.0%	92.3%	85.0%	64.7%	14.7%	0.0%
2250	337167.76±18231.69	100.0%	99.0%	96.7%	90.0%	74.3%	19.7%	0.0%
2300	308465.43±16558.97	99.0%	97.7%	96.3%	90.3%	68.0%	14.0%	0.0%

Table 5.10: Results for model: TDA-full-refining (1-ply)

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
2050	500123.64±22378.11	100.0%	100.0%	99.5%	99.5%	96.5%	46.0%	0.0%
2100	522455.76±37688.23	100.0%	100.0%	99.5%	99.5%	95.0%	49.0%	0.0%
2150	505479.60±25383.85	100.0%	100.0%	100.0%	98.5%	95.0%	44.5%	0.0%
2200	514478.80±18528.61	100.0%	100.0%	99.5%	99.0%	91.5%	49.5%	0.0%
2250	544596.52±34317.07	100.0%	100.0%	100.0%	99.0%	96.0%	58.5%	0.0%
2300	545302.60±30462.35	100.0%	100.0%	100.0%	100.0%	97.5%	53.5%	0.0%

Table 5.11: Results for model: TDA-full-refining (3-ply)

The fine-tuned TD-afterstate-full model achieves the highest mean scores among all models, demonstrating the effectiveness of generating and training with accurate data.

5.3.4 Comparison and Analysis of Method Performance

Figure 5.7 compares the mean scores of all methods from refining using 1-ply and 3-ply search strategies.

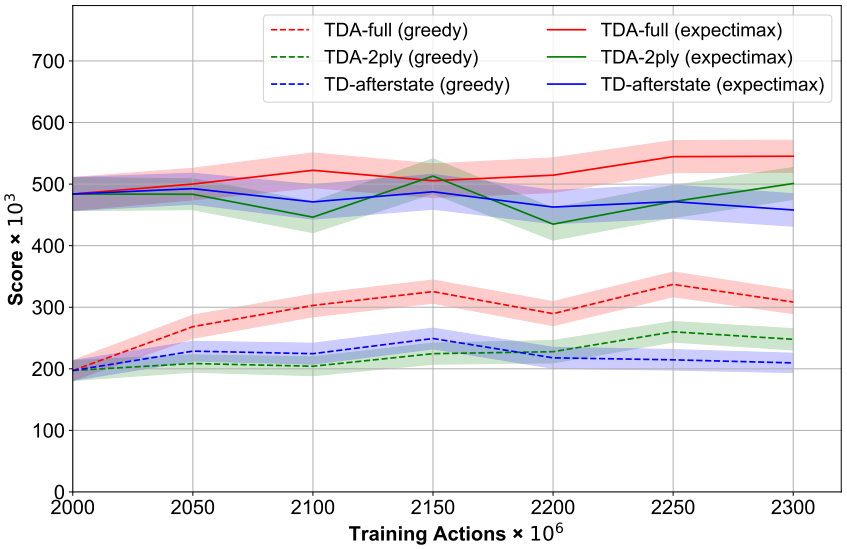


Figure 5.7: Comparison of mean scores for all methods using 1-ply and 3-ply search strategies.

The results show that the fine-tuned models consistently outperform the models trained from scratch. The TDA-full method with fine-tuning achieves the best performance, indicating that considering the full afterstate and leveraging pre-trained models significantly enhances the learning process.

5.3.5 Summary

In this section, we presented the experimental results of the methods involving generating and training with accurate data. The fine-tuning approaches demonstrate substantial improvements over training from scratch. The TD-afterstate-full method with fine-tuning achieves the highest mean scores and probabilities of reaching higher tile values, confirming the effectiveness of our proposed methods.

5.4 Experimental Results of Reducing Value Gaps

5.4.1 Performance Metrics Overview

In this subsection, we present the experimental results of the methods that involve reducing value gaps. We will show the average score performance and the average afterstate 1py value and 2-ply value. The average score for models is evaluated using 1-ply and 3-ply search strategies.

5.4.2 TDA-all Method Results

The TDA-all method is trained from the best checkpoint of TD-afterstate baseline method. Figure 5.8 shows the greedy search score with the 95% interval.

From the greedy search scores shown in Figure 5.8, it is evident that the TDA-all method fails to converge and exhibits unstable performance throughout training. This observation confirms that expanding the updates to all afterstates does not directly translate into improved gameplay quality. Instead, the method quickly collapses due to severe overestimation.

Table 5.12 shows this issue by comparing the average 1-ply afterstate values obtained from 100 games at the end of training. While the baseline TD-afterstate method produces a reasonable estimate of 1.58×10^5, the TDA-all network value is 1.12×10^{14}. This result shows that TDA-all severely overestimates the values of afterstates, leading

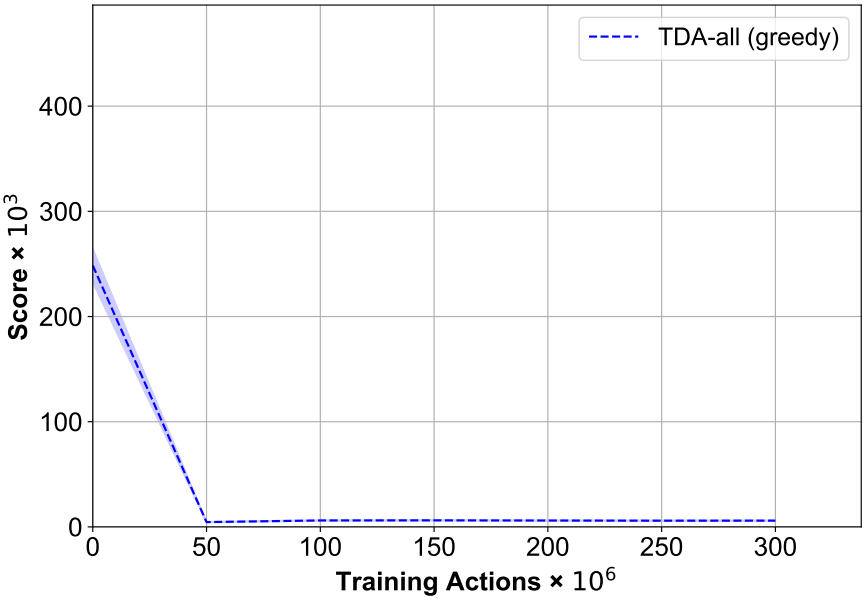


Figure 5.8: Score analysis of the TDA-all method with fine-tuning using greedy 1-ply search strategies.

to failed training outcomes and confirming the necessity of corrective strategies such as DDQN TDA-all.

Table 5.12: The network value for afterstates

	Network Value
TDA	1.58×10^5
TDA-all	$1.12 \times 10^{\mathbf{14}}$

5.4.3 DDQN TDA-all Method Results

The DDQN TDA-all method is trained from the best checkpoint of the TD-afterstate baseline method. Figure 5.9 shows the greedy search score with the 95% confidence interval.

From the greedy search scores shown in Figure 5.9, it is clear that DDQN TDA-all avoids the training failure observed in TDA-all. Instead, the method shows a stable performance, with scores gradually increasing during training and eventually reaching a plateau. This demonstrates that separating action selection and evaluation provides a more reliable basis for subsequent value analysis.

Figure 5.10 shows the differences between 2-ply and 1-ply values from afterstates of different ranks. The curves correspond to the different ranked afterstates. As shown in the figure, DDQN TDA-all successfully alleviates the value gap problem: the differences

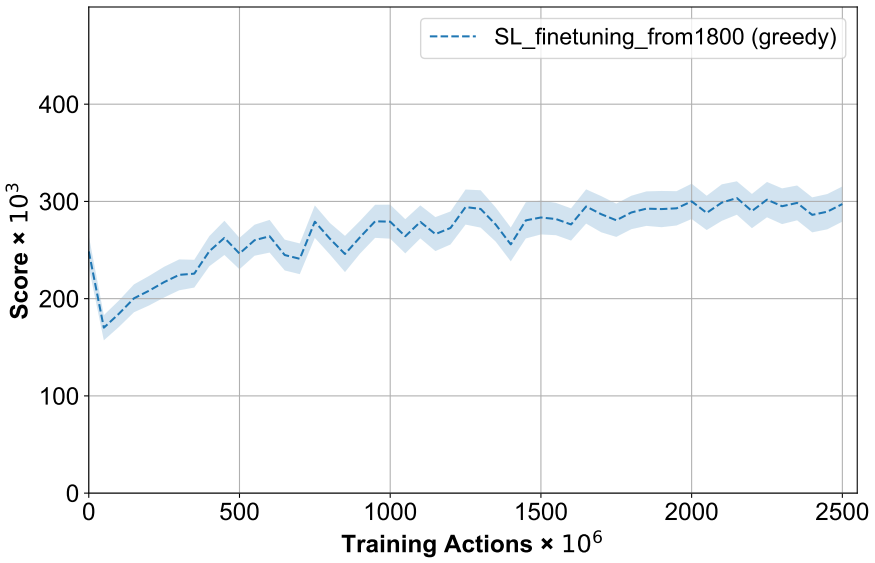


Figure 5.9: Score analysis of the DDQN TDA-all method with fine-tuning using greedy 1-ply search strategies.

between 2-ply and 1-ply values remain relatively small and stable throughout training, in contrast to the severe divergence observed in TDA-all.

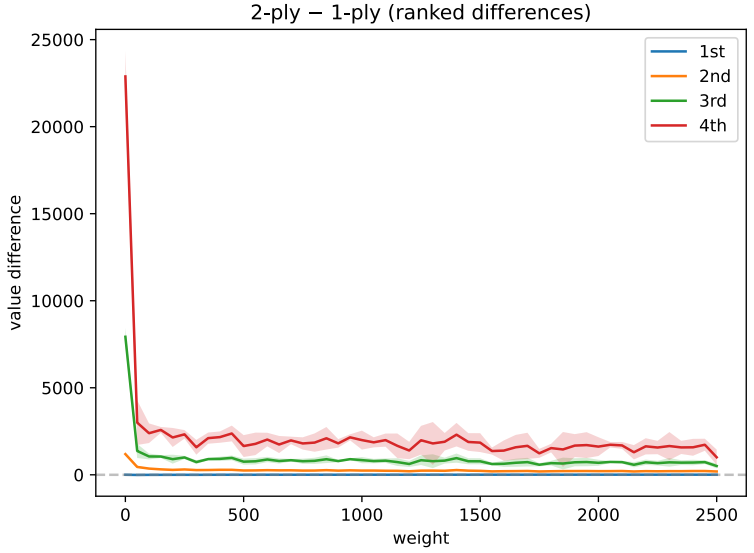


Figure 5.10: Value differences between 2-ply and 1-ply afterstates for different ranks under DDQN TDA-all training.

Figure 5.11 shows the 1-ply and 2-ply values of afterstates across different ranks during DDQN TDA-all training. Compared with the baseline TD-afterstate method, we find an initial increase in values at the early stage of training. As training progresses, the values gradually decrease and become stabilize. Importantly, no severe overestimation is observed in any rank, indicating that DDQN TDA-all successfully controls the value

escalation problem that caused TDA-all to collapse. In addition, the figure shows a clear rank-based trend: higher-ranked afterstates consistently exhibit lower overall values than lower-ranked ones, while the gap between 1-ply and 2-ply curves remains small.

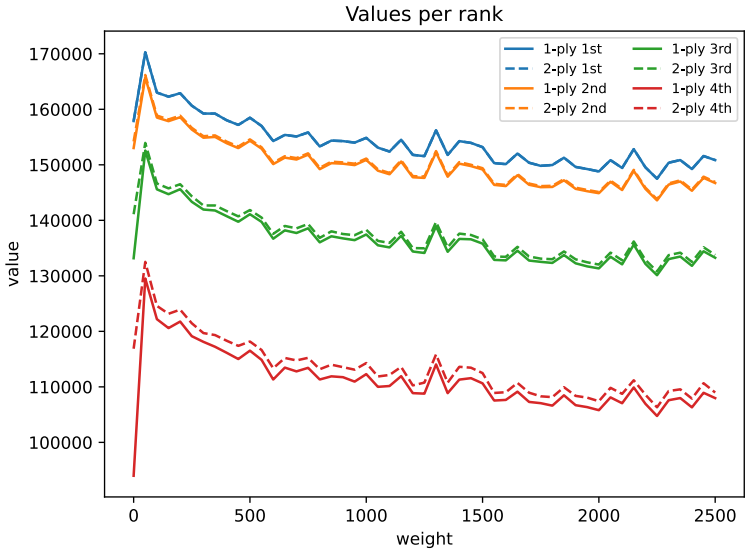


Figure 5.11: 1-ply and 2-ply values of afterstates across different ranks under DDQN TDA-all training.

5.4.4 DDQN TDA-all with TDA-full Method Results

To further examine the effectiveness of DDQN TDA-all, we selected the checkpoint with the best 1-ply score obtained from DDQN TDA-all training and continued training it under the TDA-full method. Figure 5.12, Table 5.13, and Table 5.14 present the performance of this setup, evaluated with both 1-ply greedy search and 3-ply expectimax search. The results show that the agent improves during training, achieving a maximum score of 364,083.16 under 1-ply greedy search and 563,511.42 under 3-ply expectimax search. These findings indicate that combining DDQN TDA-all with TDA-full can lead to competitive gameplay performance across different search depths.

Figure 5.13 shows the differences between 2-ply and 1-ply values for afterstates of different ranks when continuing training with TDA-full from the best DDQN TDA-all checkpoint. While the previous results shown that the refined agent achieved higher gameplay scores, this figure reveals that the value gaps increase again during TDA-full training. In particular, the differences for higher-ranked afterstates (e.g., 3rd and 4th) grow substantially as training progresses, indicating that performance improvement

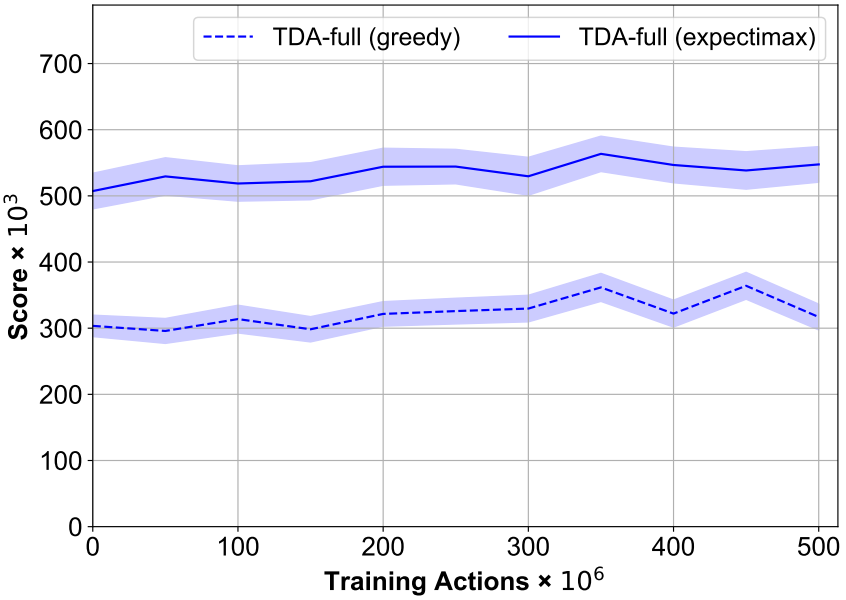


Figure 5.12: Score analysis of DDQN TDA-all combined with TDA-full.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
0	303499.24±21767.76	99.7%	99.0%	97.3%	92.3%	68.0%	12.0%	0.0%
50	295811.51±25603.30	99.3%	98.0%	96.0%	87.7%	64.7%	14.3%	0.0%
100	313761.40±38575.93	99.3%	98.3%	96.3%	87.0%	66.7%	17.0%	0.0%
150	298340.76±19677.31	98.7%	97.3%	95.3%	87.3%	65.0%	14.0%	0.0%
200	321576.51±6690.58	99.7%	99.0%	98.0%	92.0%	72.3%	16.0%	0.0%
250	325826.47±20722.84	99.0%	98.3%	95.7%	89.0%	74.3%	16.0%	0.0%
300	329643.47±14634.97	98.0%	97.7%	95.3%	89.7%	73.3%	19.0%	0.0%
350	361654.55±20111.14	99.3%	98.0%	96.3%	92.3%	75.3%	26.7%	0.0%
400	322033.37±29150.98	99.3%	98.0%	95.0%	88.3%	71.3%	18.3%	0.0%
450	364083.16±35717.59	100.0%	99.7%	99.3%	94.7%	77.7%	23.3%	0.0%
500	316883.52±26569.37	98.3%	96.3%	95.3%	89.0%	70.7%	16.3%	0.0%

Table 5.13: Results for model: DDQN TDA-all with TDA-full (1-ply)

comes at the cost of reintroducing the value overestimation problem that DDQN TDA-all initially mitigated.

5.5 Discussion

In this chapter, we have presented the experimental results of various models and methods applied to the game 2048. Our primary objective was to enhance the performance of the models by incorporating global embedding techniques and generating accurate training data. The results demonstrate that our approaches significantly improve the models' performance compared to the baseline.

weight	Mean±SD	1024	2048	4096	8192	16384	32768	65536
0	507264.26±7332.43	100.0%	100.0%	100.0%	99.5%	94.5%	49.0%	0.0%
50	529431.42±24315.57	100.0%	100.0%	99.0%	98.0%	93.5%	55.0%	0.0%
100	518683.22±30630.67	100.0%	100.0%	99.5%	99.0%	95.0%	49.5%	0.0%
150	522082.64±36717.67	99.5%	99.5%	99.5%	98.0%	94.0%	51.5%	0.0%
200	544052.58±28506.66	100.0%	100.0%	99.5%	99.5%	94.5%	55.5%	0.0%
250	544275.90±34773.35	100.0%	100.0%	100.0%	100.0%	96.0%	57.5%	0.0%
300	529597.56±14647.45	100.0%	100.0%	99.5%	99.0%	94.5%	52.5%	0.0%
350	563511.42±61028.47	100.0%	100.0%	100.0%	100.0%	98.0%	58.5%	0.0%
400	546678.04±24302.39	100.0%	100.0%	100.0%	99.5%	97.5%	53.5%	0.0%
450	538387.74±35115.10	100.0%	100.0%	100.0%	100.0%	95.0%	53.0%	0.0%
500	547542.34±30202.20	100.0%	100.0%	100.0%	99.5%	95.0%	56.5%	0.0%

Table 5.14: Results for model: DDQN TDA-all with TDA-full (3-ply)

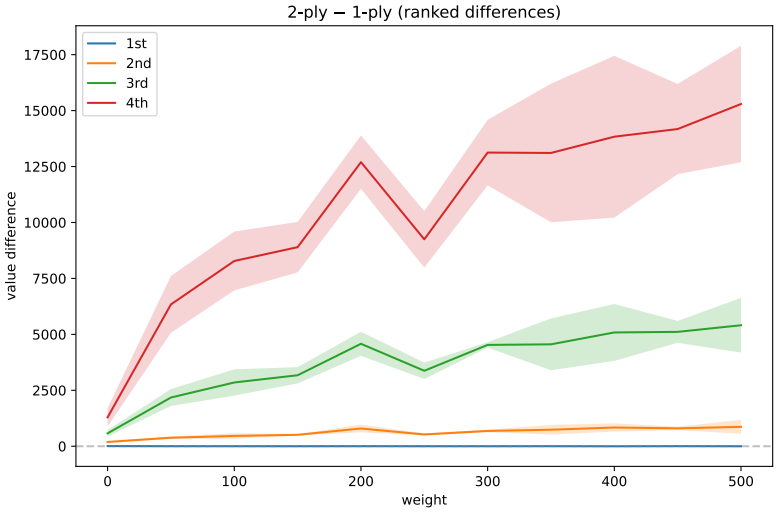


Figure 5.13: Value differences between 2-ply and 1-ply afterstates across different ranks when refining DDQN TDA-all with TDA-full.

5.5.1 Effectiveness of Global Embedding Methods

The experimental results indicate that all global embedding methods outperform the baseline CNN22 model. Specifically, models such as BP-CNN, CNN-CNN, FC-CNN consistently achieve higher mean scores during gameplay. The incorporation of global embeddings allows the models to capture more comprehensive features of the game state, leading to better decision-making and strategy formulation.

The FC-CNN model, in particular, shows the most significant improvement over the baseline with the best 1-ply score reaching 248,575.16 and the best 3-ply expectimax score reaching 486,998.62. By combining fully connected layers with convolutional neural networks, the model effectively learns complex patterns and representations of the game state. This suggests that integrating different neural network architectures can enhance the model's ability to generalize and perform better in the game.

5.5.2 Effectiveness of Refining Evaluation Function

Our experiments with the TD-afterstate-full and 2-ply TD-afterstate methods reveal that both approaches achieve better results than the standard TD-afterstate method. The fine-tuned TD-afterstate-full model attains the highest mean scores among all evaluated models. This improvement can be attributed to the generation of accurate training data, which provides the model with better quality information for learning.

By considering the full afterstate in the TDA-full method, the model gains a more detailed understanding of the potential future game states resulting from its actions. TDA-full method get the best performance with the best 1-ply score reaching 337,167.76 and the best 3-ply expectimax score reaching 545,302.60. This comprehensive perspective enables the model to make more informed decisions, leading to improved performance. Additionally, fine-tuning the models using pre-trained networks further enhances their capabilities, as it allows the models to leverage previously learned features and adapt to new data more efficiently.

5.5.3 Effectiveness of Reducing Value Gaps

Our experiments with TDA-all and DDQN TDA-all reveal clear differences in their ability to handle value estimation. While TDA-all attempts to address insufficient exploration by updating all possible afterstates, it suffers from severe overestimation and ultimately fails to converge. In contrast, DDQN TDA-all effectively mitigates this problem by separating action selection and evaluation, resulting in more stable training curves and controlled value estimates. Although DDQN TDA-all does not directly target score maximization, its success in reducing value gaps provides a more reliable foundation for subsequent refinement with methods such as TDA-full.

In addition, we further examined the effectiveness of combining DDQN TDA-all with TDA-full refinement. Starting from the best checkpoint of DDQN TDA-all, the model continued training under the TDA-full method. The results show that this combination achieves stable improvements in gameplay performance, with the best 1-ply score reaching 364,083.16 and the best 3-ply expectimax score reaching 563,511.42. These scores outperform the baseline TDA-all and are competitive with other refined models. However, value analysis also reveals that while TDA-full training enhances final performance, it reintroduces value gaps again, especially for higher-ranked afterstates. This indicates that DDQN TDA-all successfully provides a robust starting point, but

Table 5.15: Highest Score

Author	Network	Training Method	Searching Method	score ($\times 10^5$)
Antonoglou [83]	AlphaZero Network	AlphaZero Algorithm	24000 sims	5.66
Guei [4]	2-stage 8×6-tuple network	MS-OTD+TC	1-ply	3.89
			3-ply	5.28
			6-ply + D	5.85
Matsuzaki [72]	CNN22	TD-afterstate	1-ply	1.74
			3-ply	3.80
This work	FC-CNN	TD-afterstate	1-ply	2.49
			3-ply	4.87
	FC-CNN	TD-afterstate + TDA-full	1-ply	3.37
			3-ply	5.45
	FC-CNN	TD-afterstate + TDA-all (DDQN based) + TDA-full	1-ply	3.64
			3-ply	5.64

additional mechanisms are needed to fully balance both performance gains and stable value estimation during further refinement.

5.5.4 Comparison with the related method

Table 5.15 compares our proposed methods with previously reported state-of-the-art approaches. Among DNN-based 2048 players, AlphaZero [83] achieved the highest score prior to this study, while Matsuzaki's CNN22 model [72] remains the strongest open-source DNN-based baseline and serves as a key reference point in our evaluation. In contrast, Guei's MS-OTD+TC method [4], built on large n-tuple networks, represents the best-performing 2048 agent overall.

When comparing under the same search conditions, our methods demonstrate clear advantages. Using FC-CNN with TD-afterstate, our model achieves higher 1-ply and 3-ply scores than the CNN22 baseline, showing the effectiveness of our training improvements. Further refinement with TDA-full yields additional gains, surpassing Matsuzaki's results and approaching the performance of strong n-tuple agents. Finally, combining DDQN-based TDA-all with TDA-full produces the best outcome among our models, reaching a 3-ply score of 5.64×10^5, which is competitive with Guei's results and closes the gap between DNN-based approaches and n-tuple methods. These results confirm that reducing value gaps and refining evaluation functions significantly improve the effectiveness of DNN-based 2048 players.

5.6 Conclusion

In this study, we explored various techniques to improve the performance of neural network models in playing the game 2048. Our approaches focused on these aspects:

integrating global embedding methods, refining evaluation functions, and reducing value gaps.

The experimental results demonstrate that all global embedding methods except T-CNN significantly outperform the baseline CNN22 model. The enhanced models effectively capture the global structure and patterns of the game state, leading to better decision-making and higher scores. Among them, the FC-CNN model achieves the best overall performance, indicating the effectiveness of combining fully connected layers with convolutional neural networks.

Furthermore, the TDA-full and TDA-2ply methods show substantial improvements over the standard TD-afterstate approach. By generating and training with refining evaluation functions, these methods enable the models to make more informed decisions based on a comprehensive understanding of potential future states. Fine-tuning these models with pre-trained networks further enhances their performance, achieving the highest mean scores in our experiments.

In addition, our investigation of TDA-all and its DDQN-based variant shows the importance of addressing value estimation stability. While TDA-all attempts to reduce value gaps by updating all possible afterstates, it suffers from severe overestimation and quickly fails during training. By incorporating the DDQN principle of separating action selection and evaluation, DDQN TDA-all effectively mitigates this issue, resulting in stable convergence and controlled value estimates. When further refined with TDA-full, the combined approach achieves competitive performance with the best n-tuple based methods, narrowing the gap between DNN-based and traditional search-driven 2048 agents.

Overall, this work demonstrates that enhancing neural network architectures with global embeddings, refining evaluation functions, and reducing value gaps are all crucial for improving DNN-based 2048 players. Our best models not only surpass existing DNN baselines but also approach the performance of state-of-the-art n-tuple methods. These findings suggest that value stability and richer evaluation mechanisms are essential directions for future research, and they provide a solid foundation for developing more powerful deep reinforcement learning agents in 2048 and beyond.

Chapter 6

Conclusion and Future Work

6.1 Main Contributions

In this thesis, we have explored and developed advanced methodologies to enhance the performance of neural network models in playing the game 2048. Our research focused on three primary strategies: integrating global embedding techniques into the neural network architecture, refining evaluation functions through advanced temporal difference (TD) learning methods, and reducing value gaps by enhancing the exploration ability of agents. The main contributions of this work are summarized as follows:

6.1.1 Integration of Global Embedding Techniques

We introduced novel global embedding methods to augment the baseline CNN22 neural network model. By incorporating global contextual information alongside local feature extraction, we enabled the neural network to make more informed and strategic decisions. The key innovations and contributions in this area include:

- **Bypass CNN (BP-CNN):** We enhanced the input representation by concatenating positional information directly into the input layer. This allowed the network to have explicit spatial context, improving its ability to recognize and utilize spatial patterns critical for strategic decision-making in the game.
- **Fully Connected Embedding CNN (FC-CNN):** We integrated global contextual information through dedicated fully connected layers. This method effectively captured and embedded global patterns, providing the network with a holistic view of the game state, leading to more accurate valuations of afterstates and improved strategic decision-making.

- **Convolutional Embedding CNN (CNN-CNN):** We introduced additional convolutional layers dedicated to embedding global information. By leveraging the inherent strengths of convolutional operations, CNN-CNN captured complex spatial dependencies and global patterns across the entire board.
- **Transformer Embedding CNN (T-CNN):** We integrated a Transformer Encoder layer to capture complex global dependencies across the entire board. Utilizing the self-attention mechanism inherent in Transformers, T-CNN effectively modeled long-range dependencies and global interactions between tiles.

These global embedding methods significantly outperformed the baseline CNN22 model, as demonstrated by our experimental results. By providing the convolutional layers with enriched feature representations that include both local and global perspectives of the game state, our models achieved higher mean scores and increased probabilities of reaching higher tile values during gameplay.

6.1.2 Refining Evaluation Functions

We addressed the limitations of traditional TD-afterstate methods by introducing advanced training methodologies designed to refine the evaluation functions. The key contributions in this area include:

- **Full-width TD-afterstate (TD-afterstate-full):** We reduced variance in the training data by considering all possible future states from a given afterstate. By evaluating every possible next state, the agent obtained a more accurate estimate of the value of the current afterstate, leading to more stable learning and improved performance.
- **TD-afterstate with 2-ply Search (2-ply TD-afterstate):** We enhanced foresight in decision-making by incorporating deeper search strategies into the training process. By extending the search depth to two layers (plies), the agent accounted for a broader range of future scenarios, improving its ability to plan ahead and make more informed decisions.
- **Fine-Tuning Approaches:** We demonstrated that fine-tuning the advanced models based on pre-trained networks further enhanced their performance. The fine-tuned TD-afterstate-full model achieved the highest mean scores among all evaluated models.

These methods effectively addressed the high variance and shallow search issues present in traditional TD learning approaches. By generating and training with more precise and informative data, our models showed substantial improvements over the standard TD-afterstate approach.

6.1.3 Reducing Value Gaps

We investigated the problem of value gaps, which arise from discrepancies between 1-ply and 2-ply evaluations in the TD-afterstate framework. Traditional methods only update the best afterstate, leaving other possible afterstates underexplored, which leads to insufficient exploration and unstable value estimation. To address this challenge, we designed novel approaches that enhance the exploration ability of agents and improve the stability of training. The key contributions in this area include:

- **All-updated TD-afterstate (TDA-all):** We extended the TD-afterstate update rule by updating all possible afterstates instead of only the best one. This approach encouraged broader exploration and reduced the reliance on a single afterstate. However, we observed that TDA-all suffered from severe overestimation, which limited its effectiveness.

- **Double Deep Q-Network based All-updated TD-afterstate (DDQN TDA-all):** To overcome the overestimation problem of TDA-all, we introduced DDQN TDA-all, which separates action selection and action evaluation using two neural networks. This design mitigated overestimation, resulting in more stable training and controlled value estimates.

- **Combination with TDA-full Refinement:** We further combined DDQN TDA-all with TDA-full fine-tuning. This hybrid approach not only stabilized training but also achieved competitive performance compared to state-of-the-art n-tuple methods, with our best model (FC-CNN + DDQN TDA-all + TDA-full) reaching a 3-ply score of 5.64×10^5, significantly surpassing our baseline method.

These contributions show the importance of addressing insufficient exploration in Deep RL agents for 2048. By reducing value gaps and mitigating overestimation, our proposed methods enhanced both the stability and the ultimate performance of the agents.

6.2 Research Limitations

While our research has yielded promising results, there are certain limitations and factors that may influence the outcomes. Recognizing these limitations is crucial for interpreting the results accurately and for guiding future research efforts.

6.2.1 Computational Complexity

One of the primary limitations of our proposed methods, particularly the TD-afterstate-full and 2-ply TD-afterstate methods, is the increased computational complexity. Evaluating all possible next states and their corresponding afterstates requires significantly more computational resources and time. The combinatorial explosion of possible states, especially in the TD-afterstate-full method, poses practical challenges for real-time implementation and scalability.

6.2.2 Limited Exploration of Hyperparameters

Due to time and resource constraints, our experiments may not have exhaustively explored the entire hyperparameter space for the neural network models and training procedures. Parameters such as learning rates, network architectures, and training schedules can significantly impact the performance of the models. There may exist configurations that could further enhance the models' performance, which were not discovered during our experimentation.

6.2.3 Generalization to Other Games

While the methods developed have the potential to be applied to other games and sequential decision-making tasks, our research was specifically focused on the 2048 game. The effectiveness of global embedding techniques and advanced TD learning methods in other domains remains to be validated. Factors unique to other games, such as different state spaces, action complexities, and reward structures, may influence the applicability and performance of our methods.

6.2.4 Randomness in the Game Environment

The inherent randomness in the 2048 game, due to the random placement of new tiles, can still introduce variance in the results, despite our efforts to mitigate it through advanced training methods. This randomness may affect the consistency of the models'

performance across different game sessions and could influence the statistical significance of the observed improvements.

6.2.5 Evaluation Metrics Limitations

Our evaluation primarily focused on metrics such as mean scores, standard deviations, and the probabilities of reaching specific tile values. While these metrics provide valuable insights into the models' performance, they may not capture all aspects of strategic gameplay, such as the efficiency of move sequences or adaptability to different game situations.

6.3 Future Work Outlook

Building upon the findings and limitations of our research, several avenues for future work can be pursued to further enhance the performance of neural network models in the game 2048 and extend the applicability of our methods to other domains.

6.3.1 Optimization of Computational Efficiency

Addressing the computational complexity of the advanced TD learning methods is a critical area for future research. Potential approaches include:

- **Efficient Sampling Techniques:** Instead of evaluating all possible next states, smart sampling methods could be employed to approximate the expected values with reduced computational overhead.
- **Parallel Computing and Hardware Acceleration:** Utilizing parallel processing capabilities and hardware accelerators such as GPUs or TPUs can significantly reduce computation time and enable the exploration of deeper search depths.
- **Pruning Strategies:** Implementing pruning techniques to eliminate less promising branches during the search process can help focus computational resources on the most relevant states.

6.3.2 Exploration of Alternative Neural Network Architectures

Further research into different neural network architectures may yield performance improvements:

- **Deeper Networks:** Investigating deeper convolutional networks or residual networks may enhance the models' ability to capture complex patterns and dependencies.
- **Recurrent Neural Networks (RNNs):** Incorporating RNNs or Long Short-Term Memory (LSTM) networks could help model temporal dependencies and sequences of moves in the game.
- **Graph Neural Networks (GNNs):** Since the game board can be represented as a graph, GNNs might effectively capture the relationships between tiles and improve strategic planning.

6.3.3 Application to Other Games and Domains

Extending our methods to other games and sequential decision-making tasks is a promising direction:

- **Testing on Different Games:** Applying the global embedding techniques and advanced TD learning methods to games with varying complexities, such as Go, Chess, or other puzzle games, can validate the generality of our approaches.
- **Real-World Decision-Making Tasks:** Adapting the methods to real-world problems, such as robotics, resource management, or traffic control, where sequential decision-making is crucial, could demonstrate the practical applicability of our research.

6.3.4 Enhancement of Training Techniques

Improving the training procedures and exploring new techniques could further enhance model performance:

- **Curriculum Learning:** Implementing curriculum learning strategies, where the model is trained on tasks of increasing difficulty, may facilitate better learning and generalization.
- **Transfer Learning:** Leveraging pre-trained models from related tasks or games could provide a strong initialization and accelerate the learning process.

- **Reinforcement Learning Algorithms:** Experimenting with other reinforcement learning algorithms, such as Q-learning, policy gradients, or actor-critic methods, might yield additional performance gains.

6.3.5 Incorporation of Advanced Search Strategies

Combining neural network models with advanced search strategies could enhance decision-making:

- **Monte Carlo Tree Search (MCTS):** Integrating MCTS with the neural network's value estimations could improve the agent's ability to plan ahead and explore more promising move sequences.
- **Hybrid Approaches:** Developing hybrid models that combine heuristic search methods with learned value functions may balance computational efficiency with strategic depth.

6.3.6 Robustness and Adaptability Studies

Investigating the robustness and adaptability of the models under varying conditions is important:

- **Adversarial Testing:** Evaluating the models against adversarial strategies or modified game rules can assess their adaptability and resilience.
- **Uncertainty Modeling:** Incorporating uncertainty estimation methods to quantify the confidence of the model's predictions may enhance decision-making under uncertain conditions.

6.4 Closing Remarks

In conclusion, this thesis has made significant strides in enhancing neural network-based agents for the game 2048 through the integration of global embedding techniques, refining the evaluation functions, and reducing the value gaps. The proposed methods have demonstrated substantial improvements over baseline models, contributing valuable ideas to the fields of game artificial intelligence and machine learning.

While challenges remain, particularly regarding computational efficiency and generalizability, the research provides a foundation for future exploration. By addressing

the identified limitations and pursuing the suggested future work directions, further advancements can be achieved, potentially extending the benefits of these methods to a wide range of applications requiring strategic decision-making and planning.

The ongoing development and refinement of these techniques hold promise for advancing artificial intelligence capabilities, ultimately contributing to the creation of more intelligent, adaptable, and efficient systems across various domains.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. In *Nature*, volume 529, pages 484–489. Nature Publishing Group, 2016.

[3] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Karen Lai, Arthur Guez, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. In *arXiv preprint arXiv:1712.01815*, 2017.

[4] Hung Guei, Lung-Pin Chen, and I-Chen Wu. Optimistic temporal difference learning for 2048. *IEEE Transactions on Games*, 2021.

[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[6] V François-Lavet, P Henderson, R Islam, M-G Bellemare, and J Pineau. An introduction to deep reinforcement learning. In *Foundations and Trends® in Machine Learning*, volume 11, pages 219–354. Now Publishers, Inc., 2018.

[7] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[8] Arthur Guez, Robert Stengel, and Csaba Szepesvari. Investigation of the performance of the game 2048 using reinforcement learning techniques. In *Proceedings of the 24th European Conference on Machine Learning*, pages 234–247. Springer, 2015.

[9] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016.

[10] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[11] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

[12] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[13] Christopher J.C.H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, volume 8, pages 279–292. Springer, 1992.

[14] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

[15] Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. *SIAM Journal on Control and Optimization*, 38(1):94–123, 2000.

[16] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[17] Volodymyr Mnih, Adrià Puigdomènech Badia, et al. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1928–1937. PMLR, 2016.

[18] Timothy P Lillicrap et al. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

[19] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[20] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[21] Sergey Levine, Aviral Kumar, Justin Fu, and George Tucker. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[22] Peter Henderson et al. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[23] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, volume 29, pages 4026–4034, 2016.

[24] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[25] John Schulman, Philipp Moritz, Sergey Levine, Michael I Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

[26] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, 1992.

[27] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

[28] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[29] Todd Hester et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[30] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, volume 33, pages 19884–19895, 2020.

[31] Marcin Andrychowicz et al. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, pages 5048–5058, 2017.

[32] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

[33] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[34] Vinicius Zambaldi et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

[35] Junhyuk Jiang and Fei Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems*, pages 7254–7264, 2018.

[36] Carlos Guestrin, Michail G Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 227–234, 2002.

[37] Yujia Wang et al. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[38] Ashish Vaswani et al. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[39] Franco Scarselli et al. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[40] Emilio Parisotto and et al. Stabilizing transformers for reinforcement learning. *International Conference on Machine Learning*, pages 7487–7498, 2020.

[41] Jakob Foerster et al. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

[42] Sainbayar Sukhbaatar and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

[43] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. In *Discrete Event Dynamic Systems*, volume 13, pages 41–77. Springer, 2003.

[44] Pierre-Luc Bacon, Mohammad Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[45] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[46] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[47] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[48] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, 2016.

[49] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016.

[50] David Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[51] OpenAI. Openai five. *Accessed on: Oct 1, 2023. [Online]. Available: https://openai.com/five*, 2019.

[52] Oriol Vinyals et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[53] Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

[54] Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456):885–890, 2019.

[55] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International Conference on Machine Learning*, pages 2048–2056. PMLR, 2020.

[56] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

[57] Peter Henderson et al. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

[58] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in Neural Information Processing Systems*, pages 4026–4034, 2016.

[59] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A dissection of overfitting and generalization in deep reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018.

[60] Zhe Yuan, Huakun Zhang, Jianhua Zeng, and Zongben Xu. A survey on reinforcement learning for combinatorial optimization. *IEEE Transactions on Cybernetics*, 52(4):2498–2512, 2022.

[61] Dario Amodei et al. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[62] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. *arXiv preprint arXiv:2006.16712*, 2020.

[63] Rishabh Agarwal, Marlos C Machado, Pablo Samuel Castro, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34:29304–29320, 2021.

[64] Adrià Puigdomènech Badia et al. Never give up: Learning directed exploration strategies. *International Conference on Learning Representations*, 2020.

[65] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[66] Lasse Espeholt et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *International Conference on Machine Learning*, pages 1407–1416, 2018.

[67] Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[68] Gabriele Cirulli. 2048. `https://github.com/gabrielecirulli/2048`, 2014. Accessed: October 1, 2023.

[69] Marcin Szubert and Wojciech Jaśkowski. Temporal difference learning of n-tuple networks for the game 2048. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

[70] Dennis Michulke and Pei Wu. Playing 2048 with deep reinforcement learning. In *University of Freiburg, Technical Report*, 2014.

[71] Nicholas Lubberts and Risto Miikkulainen. On-line construction of n-tuple networks for games. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 635–642, 2001.

[72] Kiminori Matsuzaki. Developing value networks for game 2048 with reinforcement learning. *Journal of Information Processing*, 29:336–346, 2021.

[73] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[74] Wang Weikai and Matsuzaki Kiminori. Improving dnn-based 2048 players with global embedding. In *2022 IEEE Conference on Games (CoG)*, pages 628–631. IEEE, 2022.

[75] Wojciech Jaśkowski. Mastering 2048 with delayed temporal coherence learning, multistage weight promotion, redundant encoding, and carousel shaping. *IEEE Transactions on Games*, 10(1):3–14, 2017.

[76] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[77] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[78] Eric Van den Steen. Rational overoptimism (and other biases). *American Economic Review*, 94(4):1141–1151, 2004.

[79] James E Smith and Robert L Winkler. The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322, 2006.

[80] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8: 279–292, 1992.

[81] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23, 2010.

[82] Haobo Jiang, Jin Xie, and Jian Yang. Action candidate based clipped double q-learning for discrete and continuous action tasks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7979–7986, 2021.

[83] Ioannis Antonoglou, Julian Schrittwieser, Sherjil Ozair, Thomas K Hubert, and David Silver. Planning in stochastic environments with a learned model. In *International Conference on Learning Representations*, 2021.

List of Publication

Publication of journal:

- Refining Evaluation Functions for Game 2048 by Extended Temporal Difference Learning
  Authors: Wang Weikai, Kiminori Matsuzaki
  (conditionally accepted at the academic journal "IEEE Transactions on Games", 2025.)

Publication of conference:

- Improving DNN-Based 2048 Players with Global Embedding
  Authors: Wang Weikai, Kiminori Matsuzaki
  (Published and presented at the IEEE Conference on Games, 2022.)