Deep Q-Network Implementation for 2048 Game with Action Masking and K-Step Lookahead

January 4, 2026

Author Contributions

- Chen XinHui: 19%
- Liao WenQi: 21%
- Wang XiaoTong: 19%
- Yu GaoDi: 13%
- Ye HanWen: 13%
- Xie JingNing: 15%

Abstract

This paper presents an implementation of Deep Q-Network (DQN) for playing the 2048 game, incorporating several novel techniques including data augmentation, reward reshaping, action masking, and K-step lookahead search. We analyze the challenges of applying reinforcement learning to deterministic puzzle games like 2048, highlighting the fundamental limitations of value-based methods in environments where reasoning outperforms pattern recognition. Our findings suggest that while DQN can achieve moderate performance, traditional algorithms with Monte Carlo Tree Search (MCTS) demonstrate superior performance by exploring more game states through deliberate search rather than learning value functions.

1 Introduction

The 2048 game has emerged as a challenging testbed for reinforcement learning algorithms due to its combination of simple rules and strategic depth. As a single-player puzzle game, it presents unique challenges that differ significantly from traditional reinforcement learning benchmarks like Atari games. The game's deterministic nature, sparse rewards, and the critical importance of long-term planning make it an excellent platform for exploring the limitations of current deep reinforcement learning approaches.

In this work, we implement a Deep Q-Network (DQN) agent enhanced with several specialized techniques to address the unique challenges of 2048. Our contributions include: (1) a comprehensive data augmentation strategy that exploits the symmetrical properties of the game board, (2) a carefully designed reward function that emphasizes survival and board development over immediate score gains, (3) action masking to prevent invalid moves and improve learning efficiency, and (4) a K-step lookahead search mechanism that combines learned value functions with limited tree search.

Through extensive experimentation, we discovered fundamental limitations of value-based methods in deterministic puzzle environments. Unlike stochastic environments where pattern recognition and generalization are crucial, 2048 rewards precise reasoning and forward planning. This insight leads us to conclude that traditional algorithms with exhaustive search capabilities may be more appropriate for such domains, highlighting an important direction for future research in game-playing AI [3, 4].

2 Preliminaries

2.1 Deep Q-Network (DQN)

Deep Q-Network (DQN) is a reinforcement learning algorithm that combines Q-learning with deep neural networks to handle high-dimensional state spaces. The core idea is to approximate the optimal action-value function $Q^*(s, a)$ using a neural network parameterized by θ. The Q-learning update rule is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{1}$$

where α is the learning rate, γ is the discount factor, r_t is the reward at time t, and s_t, a_t represent the state and action at time t.

To stabilize training, DQN introduces two key innovations: (1) experience replay, which stores transitions (s_t, a_t, r_t, s_{t+1}) in a buffer and samples random minibatches for training, breaking correlations between consecutive samples, and (2) target networks, which maintain a separate network with frozen parameters for generating target values, updated periodically to reduce oscillations [1].

2.2 The 2048 Game Environment

2048 is played on a 4×4 grid where tiles with powers of two appear randomly. The player can move tiles in four directions (up, down, left, right). When tiles with the same value collide, they merge into a single tile with double the value. The goal is to create a tile with the value 2048, though play can continue beyond this point. The game ends when no valid moves remain.

The state space consists of the board configuration, typically represented as a 4×4 matrix where each cell contains a power of two (or empty). The action space consists of four discrete moves. Rewards are only received when tiles merge, with the reward equal to the value of the merged tile.

3 Proposed Methodology

3.1 Data Augmentation

Our data augmentation strategy exploits the eightfold symmetry of the 2048 game board. For each experience tuple $(s, a, r, s', \text{done})$, we generate seven additional transformed experiences by applying rotations and reflections to both the state and next state, while appropriately mapping the action. The transformations include:

- Identity transformation: $(s, a, r, s', \text{done})$
- 90° rotation: $(\text{rot90}(s), \text{map90}(a), r, \text{rot90}(s'), \text{done})$
- 180° rotation: $(\text{rot180}(s), \text{map180}(a), r, \text{rot180}(s'), \text{done})$
- 270° rotation: $(\text{rot270}(s), \text{map270}(a), r, \text{rot270}(s'), \text{done})$
- Horizontal flip: $(\text{fliplr}(s), \text{maph}(a), r, \text{fliplr}(s'), \text{done})$
- Vertical flip: $(\text{flipud}(s), \text{mapv}(a), r, \text{flipud}(s'), \text{done})$
- Main diagonal transpose: $(\text{transpose}(s), \text{mapd}(a), r, \text{transpose}(s'), \text{done})$

- Anti-diagonal flip: $(\text{flipdiag}(s), \text{mapad}(a), r, \text{flipdiag}(s'), \text{done})$

The action mappings ensure that the transformed action corresponds to the same logical move in the transformed board. For example, a left move becomes an up move after 90° rotation. This augmentation effectively increases our dataset size by a factor of eight without requiring additional environment interactions, significantly improving sample efficiency.

3.2 Reward Reshaping

The original 2048 reward function provides sparse feedback only when tiles merge, making learning difficult. We designed a comprehensive reward function that provides dense feedback emphasizing survival and board development:

$$R_{\text{enhanced}} = R_{\text{survival}} + R_{\text{emptiness}} + R_{\text{max\_tile}} + R_{\text{monotonicity}} + R_{\text{corner}} + R_{\text{merge}} \tag{2}$$

where each component is calculated as:

- **Survival Reward**: $R_{\text{survival}} = 1.0$ - A constant reward for each move, encouraging continued play
- **Emptiness Reward**: $R_{\text{emptiness}} = 0.8 \times \text{empty\_cells}$ - Rewards maintaining empty cells for future moves
- **Maximum Tile Reward**: $R_{\text{max\_tile}} = 0.5 \times \log_2(\text{max\_tile})$ - Logarithmic reward for achieving higher tiles
- **Monotonicity Reward**: $R_{\text{monotonicity}} = 0.2 \times \text{monotonic\_rows\_cols}$ - Rewards ordered tile arrangements
- **Corner Reward**: $R_{\text{corner}} = 1.0$ if $\text{max\_tile} \geq 64$ and in corner - Encourages keeping high tiles in corners
- **Merge Reward**: $R_{\text{merge}} = 0.3 \times \log(1 + \text{original\_reward})$ - Compressed merge rewards to prevent extreme values

This reward design shifts the agent's focus from immediate score maximization to long-term survival and board development, which are more aligned with achieving high scores in 2048.

3.3 Action Masking

Invalid moves are a significant issue in 2048, as approximately 25-40% of actions are invalid depending on the board state. Traditional DQN would need to learn through negative rewards that certain actions are invalid in specific states. We implement action masking to explicitly prevent the selection of invalid actions:

Algorithm 1 Action Masking in DQN

Input: state s, Q-network Q_θ, environment env
Output: action a
$valid\_actions \leftarrow$ `check_valid_actions_2048`(board)
$action\_mask \leftarrow \text{tensor}(valid\_actions)$
$q\_values \leftarrow Q_\theta(s, action\_mask)$
$masked\_q\_values \leftarrow q\_values \times action\_mask$
$a \leftarrow \arg\max(masked\_q\_values)$
return a

The action mask is a binary vector indicating which actions are valid in the current state. During both action selection and training, we multiply the Q-values by this mask, effectively setting invalid actions to negative infinity. This prevents the agent from ever selecting or learning to value invalid actions, significantly improving learning efficiency.

3.4 K-Step Lookahead Search

To address the limitations of value-based methods in planning, we implement a K-step lookahead search that combines learned Q-values with limited tree search. The algorithm evaluates each valid action by simulating K steps into the future:

The lookahead search uses Monte Carlo simulation with action selection based on the current Q-network, creating a hybrid approach that combines learning and search. During training, we use a shallower search (K=2, N=50) for computational efficiency, while during evaluation we use a deeper search (K=5, N=1000) for better performance. This approach is inspired by the success of Monte Carlo Tree Search (MCTS) in deterministic games [3, 6] and recent work combining reinforcement learning with tree search for 2048 [4, 5].

Algorithm 2 K-Step Lookahead Evaluation

Input: state s, action a, environment env, depth K, simulations N
Output: evaluation value V
$total\_value \leftarrow 0$
for $sim = 1$ to N **do**
  $board \leftarrow env.unwrapped.board.copy()$
  $sequence\_value \leftarrow 0$
  $new\_board, reward \leftarrow$ `simulate_move_2048`($board$, a)
  $sequence\_value \leftarrow sequence\_value + reward$
  if new_board is invalid move **then**
    $total\_value \leftarrow total\_value + sequence\_value$
    continue
  end if
  for $step = 1$ to $K - 1$ **do**
    $valid\_actions \leftarrow$ `check_valid_actions_2048`(new_board)
    $next\_action \leftarrow$ `select_action`(new_board, $valid\_actions$)
    $new\_board, step\_reward \leftarrow$ `simulate_move_2048`(new_board, $next\_action$)
    $sequence\_value \leftarrow sequence\_value + \gamma^{step} \times step\_reward$
    if game over **then**
      break
    end if
  end for
  $total\_value \leftarrow total\_value + sequence\_value$
end for
$V \leftarrow total\_value/N$
return V

4 Results and Analysis

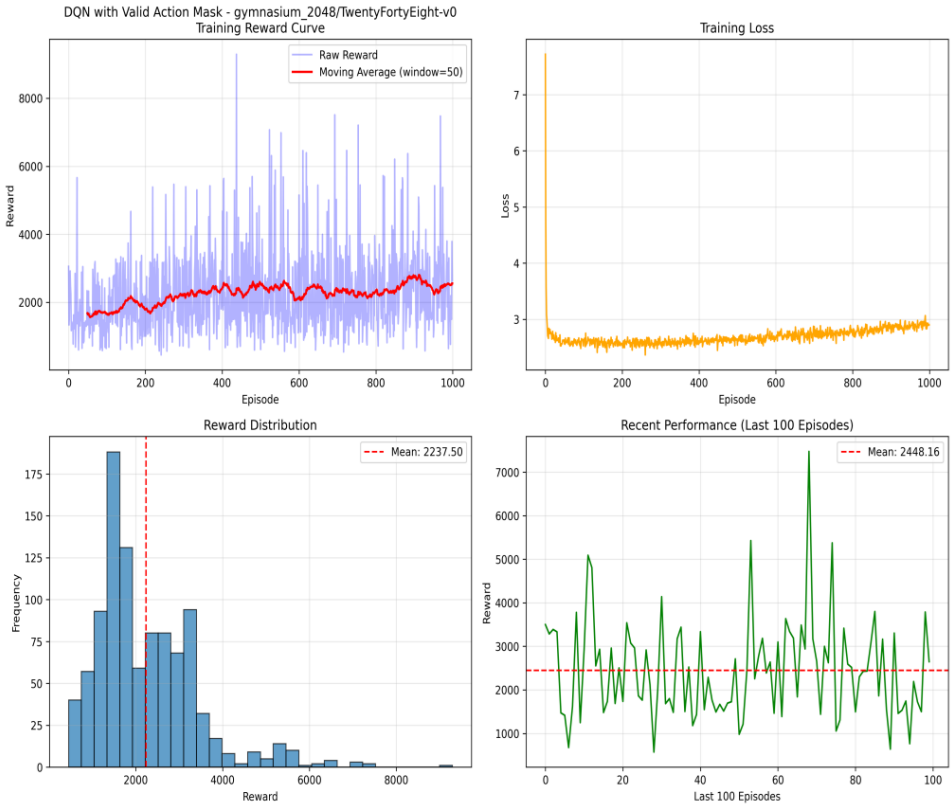

Figure 1: Training results showing reward curves, loss progression, reward distribution, and recent performance over 100 episodes

The training results reveal several interesting insights into the learning dynamics of our DQN agent for 2048. As shown in Figure 1, we observed a seemingly counterintuitive phenomenon where the training loss gradually increases while the reward also shows an upward trend. This apparent contradiction can be explained by considering the nature of the loss function as a self-checking mechanism. The increase in loss value actually indicates that the model has progressed to later stages of the 2048 game, encountering previously unseen board configurations with higher tiles such as 1024 and 512. In the context of 2048, reaching advanced game states requires first mastering the earlier stages, making this loss increase an indicator of the model's self-improvement and exploration of more complex game states.

The Reward Distribution graph provides additional insights into the model's performance consistency. The distribution shows that while the model rarely achieves very high scores, it stably obtains an average reward around 2200-2400, indicating that it has successfully mastered the basic strategy for synthesizing the 2048 tile. This consistent performance in the mid-range suggests that the agent has developed a robust understanding of fundamental game mechanics but struggles to achieve exceptional scores that require more sophisticated multi-step planning.

The Recent Performance graph, which displays the last 100 episodes, reveals significant fluctuations in the model's performance. This volatility can be attributed to several factors. First, in a deterministic environment like 2048, the epsilon-greedy exploration strategy proves inefficient, leading to unstable decision-making as the agent randomly explores actions that can be catastrophic in a game where precision is crucial. Second, the model demonstrates limited generalization capability for complex situations requiring multi-step precise planning, such as achieving the 4096 tile. The sharp peaks and valleys in the performance graph highlight the agent's inconsistency when faced with novel high-level strategic decisions, suggesting that the current approach may have reached a performance ceiling in terms of strategic depth.

Figure 2: Performance comparison between our DQN agent and human players in two different game sessions

To further evaluate our agent's performance, we conducted comparative studies against human players. As shown in Figures 2, the DQN agent demonstrates consistent gameplay patterns compared to human strategies. The agent shows more conservative move selection, prioritizing board survival and maintaining empty cells, while human players tend to take more aggressive approaches that can lead to either higher scores or earlier game termination. This comparison highlights the fundamental difference in decision-making between learned heuristics and human intuition in puzzle games.

These observations collectively suggest that while our enhanced DQN approach successfully achieves baseline competency in 2048, it faces inherent limitations in mastering the deeper strategic elements of the game that distinguish good players from exceptional ones.

5 Conclusion

Through our implementation and experimentation with DQN for 2048, we have gained several important insights into the application of deep reinforcement learning to deterministic puzzle games. While our enhancements—including data augmentation, reward reshaping, action masking, and K-step lookahead—improved upon baseline DQN performance, we discovered fundamental limitations of value-based methods in this domain.

Our key finding is that 2048 is fundamentally a game of "reasoning" rather than "understanding." Unlike stochastic environments where pattern recognition and generalization are paramount, 2048 rewards precise forward planning and logical deduction. The deterministic nature of the game means that given sufficient computational resources, traditional search algorithms can explore the game tree more thoroughly than any learned value function can approximate.

A critical challenge we encountered with DQN is the epsilon-greedy exploration dilemma. When epsilon is too large, the agent frequently makes catastrophic errors that end the game prematurely, preventing it from ever reaching high-scoring states where meaningful learning can occur. When epsilon is too small, the agent fails to explore sufficiently and gets stuck in local optima. This problem is particularly acute in 2048 because a single wrong move can irreversibly doom a game, unlike in many other reinforcement learning environments where recovery from mistakes is possible.

Our experiments suggest that traditional algorithms combined with Monte Carlo Tree Search (MCTS) achieve superior performance by systematically exploring more game states and calculating exact values rather than ap-

proximating them through neural networks. The ability of MCTS to allocate more computational resources to critical positions and to precisely calculate the consequences of specific move sequences makes it inherently better suited to deterministic puzzle games [3, 4]. This finding is consistent with recent research showing that MCTS-based approaches can achieve average scores exceeding 500,000 in 2048 [5], significantly outperforming pure reinforcement learning methods.

These findings highlight an important direction for future research: developing hybrid approaches that combine the pattern recognition capabilities of deep learning with the precise planning abilities of traditional search algorithms. Such approaches might use neural networks to guide search toward promising regions of the game tree while relying on exact calculation for final move selection, potentially achieving the best of both worlds.

In conclusion, while deep reinforcement learning has revolutionized many areas of artificial intelligence, deterministic puzzle games like 2048 reveal important boundaries where traditional algorithms still reign supreme. Understanding these boundaries is crucial for developing more robust and general game-playing AI systems that can adapt their strategies to the specific characteristics of different game domains.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533.

[2] Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence* (Vol. 30, No. 1).

[3] Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 1-43.

[4] Szubert, I., & Jaśkowski, W. (2014). Temporal difference learning of n-tuple networks for the game 2048. In *IEEE Conference on Computational Intelligence in Games* (pp. 1-8).

[5] Wu, I., Chen, H., & Shan, Y. (2022). Enhancement of CNN-based 2048 Player with Monte-Carlo Tree Search. *IEEE Transactions on Games*, 14(4), 654-663.

[6] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.

[7] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, 550(7676), 354-359.