Applying Reinforcement Learning to the Chrome Dino Run Game

Emre Iyigün, *Ilgar* Korkmaz, *Nik* Yakovlev, and *Aref* Hasan

Abstract. This paper explores the application of Reinforcement Learning (RL) techniques, specifically Double Deep Q-Network (DDQN), to the Chrome Dino Run game. The game, a simple yet challenging offline browser game developed by Google, serves as an ideal testbed for evaluating RL algorithms due to its dynamic and high-frequency action requirements. The study implements a baseline heuristic policy and investigates various DDQN variants to optimize the agent's performance. Key modifications include different learning rates, observation decay rates, and prioritized experience replay. The results demonstrate that all DDQN variants outperform the baseline, with the highest maximum score achieved by a variant with adjusted observation decay rates. However, all variants still fall short of human expert performance, indicating the need for further hyper-parameter tuning and enhanced computational resources.

1 Introduction

Reinforcement Learning (RL) has demonstrated considerable success across a range of domains, including game playing, robotics, and autonomous systems. In particular, the application of RL in strategy-based games, such as Total War and role-playing games (RPGs), has become a highly promising research area with significant potential to advance artificial intelligence (AI) and machine learning (ML). These games present unique challenges, such as managing a high-dimensional state space and controlling millions of virtual units, making them an ideal testbed for evaluating and refining RL algorithms.

The Chrome Dino Run game, also known simply as Dino, is a simple yet challenging single-player browser game developed by Google and built into the Google Chrome web browser, which activates when the browser is offline. The game environment consists of a running dinosaur that must jump over obstacles, such as cacti and birds, to survive. The actions available to the agent are jumping and ducking, with the objective being to maximize the survival time and, consequently, the score. Also, an indefinite number of states are possible due to the infinite canvas of obstacles appearing at varying distances. As shown in Figure 1, the Chrome Dino Run game appears when there is no internet connection.

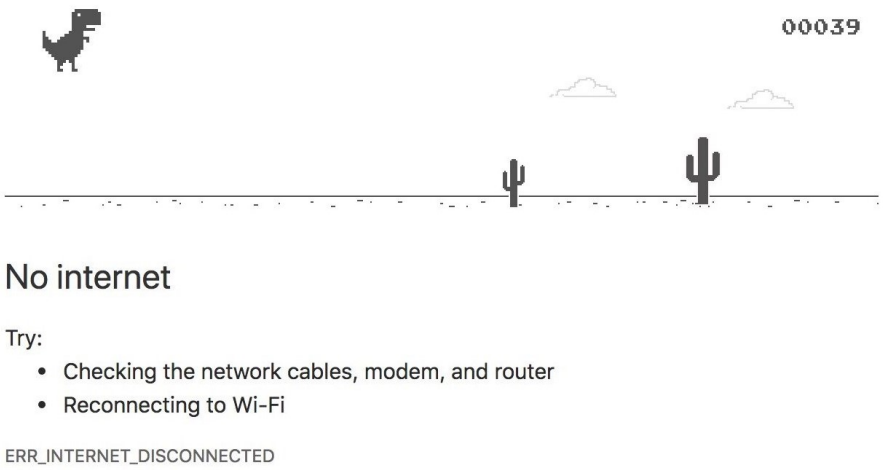

Figure 1. The Chrome Dino Run game, featuring the dinosaur on a desert background with a 'No internet' message. The game activates when there is not internet connection.

The objective of this paper is to implement a simple baseline and subsequently explore various variants of the Double Deep Q-Network (DDQN) algorithm. Utilizing the Selenium Python library, all processes are automated to ensure consistent experimentation. The primary goal is to investigate the performance of different DDQN variants and identify the most effective variant. This exploration aims to provide insights into the nuances of the DDQN algorithm and its potential optimizations for improved performance.

2 Related Work

In the following section we are going to look at papers which used Deep Q-Network (DQN) and DDQN for gaming scenarios. This section should give the reader of this paper an overview over previous work in this area.

2.1 Playing Atari with Deep Reinforcement Learning

One influential paper in deep reinforcement learning is "Playing Atari with Deep Reinforcement Learning" by Mnih et al. (2013) [2]. This work introduced a DQN capable of learning control policies from high-dimensional sensory input.

It's key contributions include:

1. End-to-End Learning: The network learned directly from raw video inputs without handcrafted features.
2. Experience Replay: This technique improved training stability by storing and randomly sampling past experiences.
3. Generalization Across Games: The same network architecture and hyperparameters were applied to seven different Atari games, outperforming previous methods on six and surpassing human experts on three.

This paper demonstrated the potential of combining deep learning with reinforcement learning, significantly advancing the field and influencing a variety of applications beyond gaming.

2.2 Comparison of Reinforcement Learning in game AI

The study [4] compared the performance of three reinforcement learning algorithms—DQN, ACER, and PPO—across 19 Atari games to evaluate their strengths and weaknesses. The results showed that the DQN algorithm achieved the best results overall, excelling in 11 out of 19 games. DQN's ability to maximize scores was due to its focus on long-term rewards, making it particularly effective in complex game environments. However, it also exhibited a larger standard deviation, indicating some instability in performance.

In contrast, ACER and PPO demonstrated more stable but generally lower scores. ACER was consistent in games requiring immediate rewards but struggled in scenarios needing long-term strategy. PPO, while stable, also tended to focus on short-term rewards, performing well in simpler games but falling short in more complex environments [4].

The experiment revealed that DQN is the most effective algorithm for maximizing scores, despite its occasional instability. On the other hand, ACER and PPO offer more consistent performance but may not reach the high scores that DQN can achieve [4].

2.3 Deep Q-Network, Double DQN, and Dueling DQN: Advancements in Reinforcement Learning

The paper by Mohit Sewak [3] explores significant advancements in Reinforcement Learning through DQN and its variants, Double DQN and Dueling DQN. These models underpin applications like AlphaGo and represent steps toward achieving General Artificial Intelligence. The paper highlights the transition from task-specific reinforcement learning agents to more general models capable of excelling in various domains.

DQN uses deep convolutional neural networks to learn policies from high-dimensional inputs, such as images from Atari games. However, it suffers from overestimation bias because the same network selects and evaluates actions, leading to overly optimistic value estimates [3].

DDQN addresses this by decoupling action selection from evaluation. It uses the online network to select actions and a separate target network to evaluate them. This separation reduces overestimation of Q-values, resulting in more stable and accurate learning. This improvement is crucial in large state-action spaces, enhancing the agent's reliability and performance [3].

3 Methods

The foundational conditions applied to all experiments are as follows:

- To set up the game environment locally, the t-rex-runner repository [5] is used and configured to run offline
- The game is configured with no acceleration and no birds for simplicity
- Jumping longer by holding the space button is also not part of the simplified game
- Only two actions are considered: jump and do nothing. Although the game includes a third action (duck) to evade birds when acceleration is present, this action is omitted in our simplified setup
- The reward system is defined as: hitting an obstacle: -1, otherwise: +0.1
- Selenium is utilized to automatically control the agent in the Chrome Browser
- The testing process will be conducted over 20 episodes using a random environment

3.1 Baseline

The baseline serves as a minimum performance threshold that any RL algorithm should exceed to be considered effective.

For the baseline, a simple heuristic policy is used where the agent only performs the "jump" action every 0.1 seconds. This frequent jumping simulates the effect of the spacebar being constantly pressed, allowing for a clear measurement of the reward and game progress achieved by this straightforward strategy.

Future algorithms can be evaluated against this baseline to ensure they provide a meaningful improvement in terms of the agent's ability to play the Dino Chrome game.

3.2 DDQN

DDQN, a value-based off-policy algorithm, is an enhancement of the traditional DQN designed to address the overestimation bias inherent in Q-learning algorithms. This bias arises when the same network is used for both action selection and action evaluation, leading to overly optimistic estimates of Q-values. DDQN mitigates this issue by decoupling these tasks, using two separate networks for action selection and evaluation. This separation provides more stable and accurate value estimates, improving the overall performance of the algorithm [1].

DDQN Algorithm Steps [1]:

1. **Initialization:**
   - Initialize two neural networks: the online network Q_θ and the target network Q_{θ^-}
   - Initialize the replay buffer D to store experience tuples (s, a, r, s')
2. **Action Selection:**
   - At each time step t, select an action a_t using an epsilon-greedy policy based on the Q-values from the online network Q_θ
3. **Experience Storage:**
   - Execute the action a_t in the environment to obtain the reward r_t and the next state s_{t+1}
   - Store the transition (s_t, a_t, r_t, s_{t+1}) in the replay buffer D
4. **Experience Replay:**
   - Randomly sample a mini-batch of transitions (s_i, a_i, r_i, s_{i+1}) from the replay buffer D
   - Compute the target Q-value y_i using the target network Q_{θ^-}:

$$y_i = r_i + \gamma Q_{\theta^-}(s_{i+1}, \arg\max_{a'} Q_\theta(s_{i+1}, a'))$$

   - Update the online network Q_θ by minimizing the loss:

$$L(\theta) = \mathbb{E}_{(s_i, a_i, r_i, s_{i+1}) \sim D}\left[(y_i - Q_\theta(s_i, a_i))^2\right]$$

5. **Target Network Update:**

   - Periodically update the target network Q_{θ^-} to match the weights of the online network Q_θ

The DDQN is implemented as a two Convolutional Neural Network (CNN), representing the Q-Network and Target Network. Each consists of three convolutional layers that extract features from a stack of four images taken from the screen using OpenCV while the game is in play mode. The images are reduced to 80x80 pixels and converted to grey scale. The convolutional layers are subsequently followed by a flattening layer and a dense (fully connected) layer. The final output layer produces two Q-values, one for each possible action (jump and do nothing). This architecture is illustrated in Figure 2.
The model is trained starting from an initial state with no predefined action. The agent observes for a specified number of steps, storing its experiences in the Replay Memory. A batch of experiences is then sampled from this memory for training. Upon the agent's death, the game is restarted. Furthermore, the Adam optimizer is employed to determine adaptive learning rates. All DDQN variants are trained for 2500 episodes, which equates to approximately five hours of training using an Intel i5 processor with 16 GB of RAM. The evaluation metrics used for training include loss values calculated using mean square error and the scores achieved at the end of each episode.

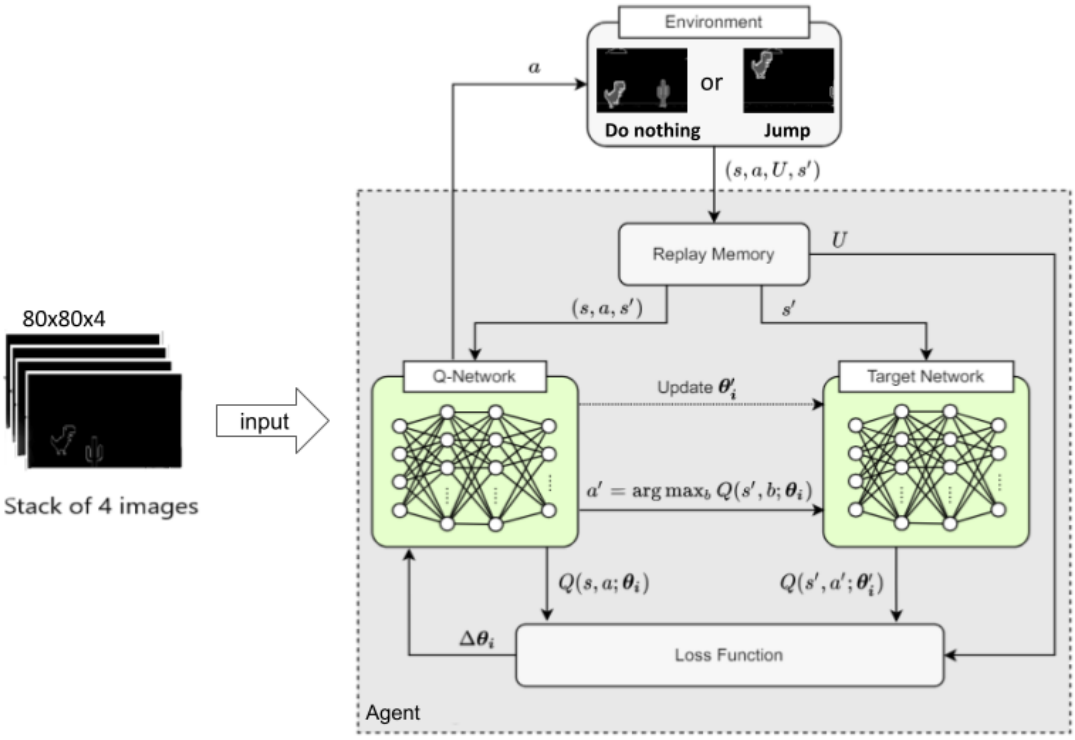

Figure 2. The diagram illustrates the process of training the DDQN. The agent receives a stack of 4 images (80x80 pixels each) representing the current state and uses its Q-Network to predict Q-values for possible actions ("Do nothing" or "Jump"). Based on these Q-values, the agent selects an action and interacts with the environment, resulting in a new state and reward. The transition is stored in the replay memory. The target network computes the target Q-value for the new state, and the Q-Network is updated by minimizing the loss function, which is the difference between the predicted and target Q-values. This process repeats to improve the agent's policy, guided by experiences stored in the replay memory. Source: Based on [6].

3.3 Why DDQN is Suitable for Chrome Dino Run?

The Chrome Dino Run game presents unique challenges that make the DDQN an ideal choice for training an agent. Firstly, DDQN effectively addresses the issue of overestimation bias, which is particularly important in a game like Chrome Dino Run where quick decision-making and high-frequency action selection are critical. The dual-network approach of DDQN ensures that action selection and evaluation are decoupled, leading to more accurate value estimates and thus, more reliable action choices [1].

Moreover, the game features dynamic and varied obstacles, requiring the agent to continuously adapt its strategies. DDQN provides a stable learning process through the use of experience replay and peri-

odic updates to the target network, which helps the agent adapt smoothly to new scenarios without being destabilized by recent experiences [3].

Additionally, studies have demonstrated that DDQN outperforms traditional DQN in various gaming environments by achieving higher and more consistent scores [1]. This enhanced performance is crucial for the Chrome Dino Run game, where the agent needs to maintain high performance over increasingly difficult obstacles.

4 Results

The following DDQN variants have been explored:

- Variant 1 (Initial Configuration)
  - Decay Rate of Past Observations: 0.99
  - Observation Timesteps Before Training: 500
  - Final Value of Epsilon (exploration probability): 0.1
  - Initial Value of Epsilon (initial randomness): 1
  - Epsilon Decay Rate: 0.9999925
  - Replay Memory Size: 1000 (number of previous transitions to remember)
  - Minibatch Size: 32
  - Learning Rate: 1e-4
  - Weight Decay for Regularization: 1e-4
  - Sync Frequency between Q_θ and Q_{θ^-}: 1000 experiences
  - Training Frequency: every 3 steps
  - Prioritized Replay: False
  - Gradient Clipping: 10 (to prevent gradient explosion)
- Variant 2 (Training with Prioritized Experience Replay)
  - Prioritized Replay: True
  - all other hyper-parameters remain the same
- Variant 3 (Training with Increased Learning Rate)
  - Learning Rate: 5e-4
  - all other hyper-parameters remain the same
- Variant 4 (Training with Decay Rate of Past Observations)
  - Decay Rate of Past Observations: 0.95
  - all other hyper-parameters remain the same

The results indicate that Variant 4 achieved the highest maximum score, outperforming the baseline and all other variants. Figure 3 shows an overall comparison of maximum scores achieved by different methods over 20 episodes.

To provide a comprehensive comparison, Table 1 presents maximum score, average score, standard deviation (Std), and median score across 20 rounds for a human expert, baseline, and the different DDQN variants.

Table 1. Performance metrics for 20 rounds. The different model variants are compared with the baseline and games completed by a human expert.

Method	Max. Score	Avg. Score	Std	Median Score
Human Expert	1399	922	324	850
Baseline	65	51	22	40
Variant 1	137	52	25	43
Variant 2	141	65	23	63
Variant 3	142	61	15	51
Variant 4	162	58	26	62

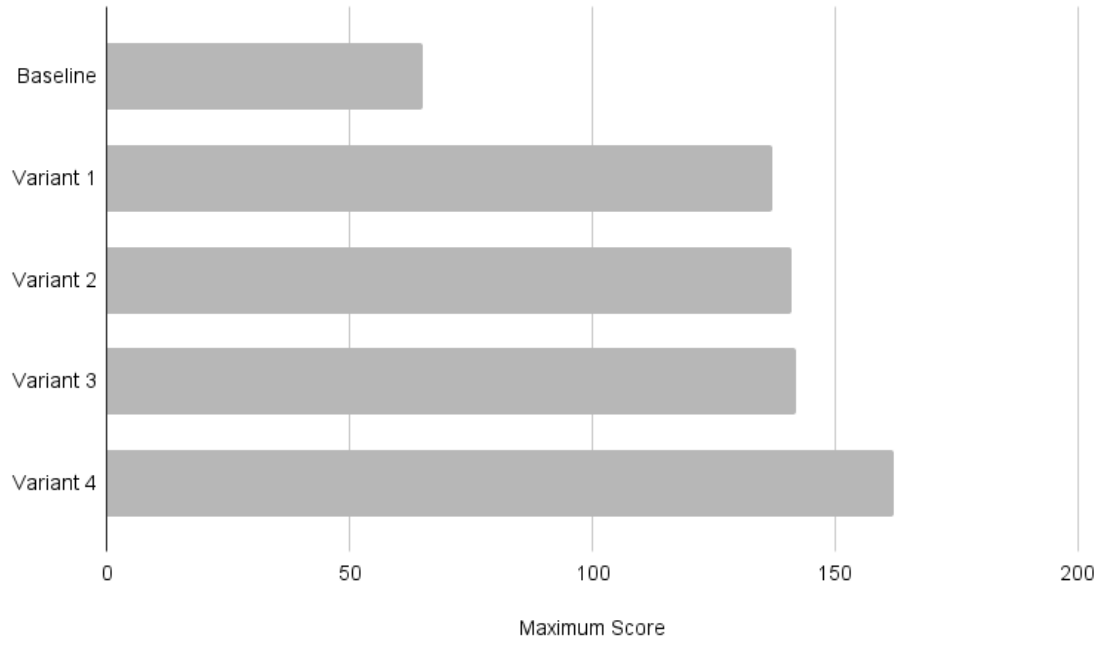

Figure 3. The chart compares the maximum scores reached by different methods. Variant 4 achieved the highest maximum score, outperforming the baseline. Variants 1, 2, and 3 also surpassed the baseline, with Variant 3 performing slightly better than Variants 1 and 2.

Variants 2 and 3 demonstrated more consistent performance with lower standard deviations, whereas the baseline and Variant 4 had higher standard deviations, indicating more variability in their performance.

Increasing the learning rate in Variant 3 resulted in a slight improvement in the maximum score compared to Variant 1, suggesting that the model benefited from a faster learning rate.

Despite improvements, all DDQN variants fell short of the human expert's performance, particularly in achieving higher maximum and average scores. This suggests that further tuning of hyperparameters and potentially more advanced training techniques are necessary to match human-level performance.

5 Conclusion

This study explored various DDQN configurations to enhance the performance of the agent in the Chrome Dino Run game. Four distinct variants were tested, each modifying key hyper-parameters to assess their impact on the model's performance. The results indicate that all DDQN variants outperformed the baseline model, where Variant 4 achieved the highest maximum score. Still all DDQN variants are far from the achieved scores by the human expert especially due the limited computing power available.

For future work, these models should be trained on a GPU-based system with increased number of episodes to obtain more significant results. Additionally, algorithms such as Deep Deterministic Policy Gradient, Dueling DQN or Policy-based RL algorithms should be implemented and extensively compared with the current results. Also exploring more sophisticated exploration strategies, such as adaptive epsilon decay or intrinsic motivation techniques, could enhance the agent's learning process.

Another important consideration is the increasing speed of obstacles with the score. To address this challenge, the game could be divided into stages, with each stage receiving input from the previous one but being trained individually. This staged approach may improve the performance of reinforcement learning algorithms and reduce score variance, leading to more robust and consistent outcomes.

Overall, a comprehensive evaluation of the hyper-parameters should be conducted to identify the optimal configurations for the model. This includes systematically testing a wide range of values for each hyperparameter to understand their impact on performance and stability. Such an approach would help in fine-tuning the model for better performance in various environments.

References

[1] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," arXiv, vol. 1509.06461, Dec. 2015, [Online]. Available: http://arxiv.org/abs/1509.06461. doi: 10.48550/arXiv.1509.06461 .

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv, Dec. 2013. [Online]. Available: https://arxiv.org/abs/1312.5602.

[3] M. Sewak, "Deep Q Network (DQN), Double DQN, and Dueling DQN: A Step Towards General Artificial Intelligence," in *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*, Jun. 2019, pp. 95-108. doi: 10.1007/978-981-13-8285-7-8. ISBN: 978-981-13-8284-0 .

[4] J. Nogae, K. Ootsu, T. Yokota and S. Kojima, "Comparison of reinforcement learning in game AI," 2022 23rd ACIS International Summer Virtual Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD-Summer), Kyoto City, Japan, 2022, pp. 82-86, doi: 10.1109/SNPD-Summer57817.2022.00022 .

[5] wayou, "t-rex-runner," GitHub repository, 2024. [Online]. Available: https://github.com/wayou/t-rex-runner

[6] Q. Vinh Do and I. Koo, "Deep Reinforcement Learning Based Dynamic Spectrum Competition in Green Cognitive Virtualized Networks," in IEEE Access, vol. 9, pp. 52193-52201, 2021, doi: 10.1109/ACCESS.2021.3069969 .