How to Teach a Plumber? Reinforcement Learning in Super Mario Bros.

Noam Benkler

Abstract

This paper provides a thorough exploration of reinforcement learning (RL) techniques within the iconic Super Mario Bros video game environment, employing various algorithms and training curricula. Focused on Double Deep Q-Networks (DDQN) and Actor-Critic methods, we investigate the impact of algorithmic enhancements, such as adding a recurrent Long Short-Term Memory (LSTM) layers and two distinct training curricula on DDQN agent performance. Our experimental results illustrate the benefits of tailored neural network design and emphasize the importance of careful curriculum construction. Through this work, we hope to offer valuable insights for both the gaming and broader RL research communities.

1 Introduction

Reinforcement Learning (RL) is a machine learning paradigm where an agent learns through interactions with its environment, refining decision-making over time based on received rewards or punishments. In complex scenarios, this trial-and-error approach proves effective. Deep Reinforcement Learning (Deep RL) takes RL further by incorporating deep neural networks to handle high-dimensional state spaces, allowing agents to learn hierarchical and abstract data representations, particularly beneficial in challenging tasks [14].

Two main categories within DeepRL are value-based methods, exemplified by Double Deep Q-Networks (DDQN), and actor-critic methods, such as Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO). Value-based methods excel in stability and simplicity, while actor-critic methods offer advantages in sample efficiency and policy optimization [11, 10].

The evaluation of RL-based agents has often been conducted in video game environments, serving as benchmarks for agent intelligence progress [6]. In this project, we leverage OpenAI Gym's Super Mario Bros. environment [5] to train an RL agent to navigate the game's first level. Building upon a baseline DDQN approach by Yuansong Feng et al.[3], we explore various algorithmic enhancements, including integrating a recurrent Long Short-Term Memory (LSTM) layer for active vision and implementing a pair of naive curricula for model training. Additionally, we introduce two competing actor-critic models to assess how these enhancements bridge the gap between value-based and actor-critic approaches. Through our experiments, we seek to develop a deeper understanding of how adding recurrence and various curricula can impact a DDQN agent's learning and performance in a video game environment.

2 Background

2.1 Deep Q-learning and DDQN

Deep Q-learning, a variant of Q-learning, addresses Markov Decision Processes (MDP) by learning action values from experiences [14]. At each timestamp t, the agent receives state S_t and reward R_t from its environment, selecting action A_t in response. This iterative process continues until reaching a terminal state, concluding the episode. The overarching goal is to maximize the return G_t, the sum of discounted rewards, by learning the value function $Q(S_t, A_t)$ through a deep neural network.

A Deep Q-network (DQN) is a multi-layered neural network mapping R^n to R^m for an n-dimensional state space and m actions [10]. DQNs may suffer from overestimation bias, leading to the development of the Double Deep Q-Network (DDQN) by Hasselt et al. [15]. The DDQN architecture addresses overestimation bias by incorporating two networks—an Online network Θ for acting and a Target network Θ' for computing the loss function, updated at a lower fixed rate.

2.2 Actor-Critic Methods

Actor-critic methods blend policy-based and value-based approaches to optimize the agent's policy in an environment [9, 13]. The actor component utilizes a neural network as a policy to output action probabilities based on the current state. Simultaneously, the critic network estimates the expected future reward by considering the current state and actor's output. An advantage function guides the actor's policy, indicating which actions lead to better outcomes. Both the actor and critic continuously update and improve through backpropagation of the advantage function, resulting in an optimal policy that maximizes expected future rewards.

Two notable algorithms in this category are the Asynchronous Advantage Actor-Critic (A3C) [9] and Proximal Policy Optimization (PPO) [13]. A3C is a parallel, asynchronous, multi-threaded implementation of the actor-critic algorithm where multiple agents are trained in parallel in their own environment, exploring different parts of the state spaces simultaneously. Each agent calculates independent policy gradients and periodically sends updates to a global network. The global network then propagates new weights to the agents at each update to ensure they share a common policy. PPO, on the other hand, uses multiple epochs of stochastic gradient ascent to perform each policy update. It improves policy training stability by limiting changes to the policy, avoiding too-large updates at each training epoch.

2.3 Approaches to Temporal Integration in DeepRL

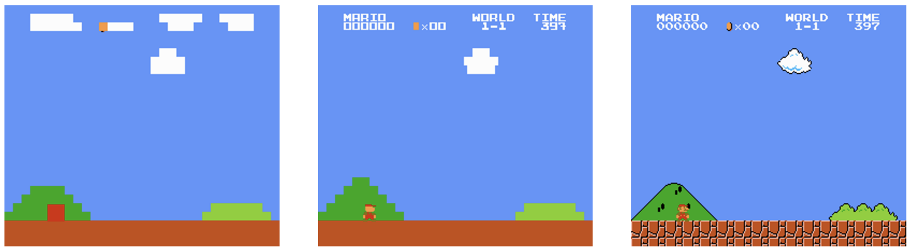
Markov Decision Processes (MDPs) represent simple environments where a state provides all necessary information for optimal agent action [14]. Real-world problems often deviate from MDP standards, leading to Partially Observable MDPs (POMDPs) [4]. To address partially observable worlds, neural agents benefit from temporal integration of observations. This involves enabling the agent to consider information over time, crucial for realistic tasks. In reinforcement learning, temporal integration can be achieved using various methods. The baseline approach, inspired by DeepMind's strategy in playing Atari games [4], involves stacking frames. However, this method has limitations. Recurrent Neural Networks (RNNs) offer a solution by directly incorporating temporal integration into the agent. Unlike stacking frames, RNNs allow the network to maintain a hidden state, capturing temporal patterns [8]. This integration is crucial for scenarios where events from earlier frames influence current decisions. The resulting agent, known as Deep Recurrent Q-Networks (DRQN), utilizes an RNN to enhance its ability to handle partially observable environments [4].

2.4 Curriculum Learning

Curriculum learning, introduced by Bengio et al. [1], draws inspiration from human learning processes. In their paper "Curriculum Learning," Bengio et al. highlight the efficacy of presenting information in a meaningful order to enhance learning, mirroring human and animal learning patterns. They propose ordering training samples from easy to hard, allowing models to grasp simpler features before tackling more complex ones. The essence of curriculum learning involves optimizing the learning process by training on a sequence of tasks, gradually increasing difficulty rather than solely focusing on the true task of interest. Various curriculum learning methods have proven beneficial in enhancing the efficiency of training DeepRL algorithms [7]. In the realm of reinforcement learning in video games, curriculum learning entails a gradual increase in task complexity (such as increasing levels, game state representation complexity, or game action space size).

3 Environment

The OpenAI Gym Super Mario Bros environment stands as a widely used and challenging platform for the evaluation and development of reinforcement learning algorithms [5]. Part of the Gym toolkit [2], this environment enables researchers and developers to train and assess agents within the iconic Super Mario Bros game, presenting a dynamic and immersive backdrop for experimentation in reinforcement learning.



A. Rectangle ROM B. Pixelated ROM C. Standard ROM

Figure 1: Mario ROM versions from least information (left, A) to most information (right, C)

3.1 Overview

Game Setting: Super Mario Bros is a classic side-scrolling platform game where the player assumes control of Mario, a plumber navigating through levels replete with obstacles, enemies, and power-ups.

Worlds: The game is partitioned into multiple worlds, each containing several stages.

Observation Space: The environment furnishes a 3x224x256 RGB image as the observation space, enabling the agent to perceive the current game state.

Action Space: The action space comprises discrete actions such as movement (left or right), jumping, and other gameplay-related actions like crouching or sprinting.

3.2 Worlds, Stages, Versions

The game unfolds across two distinct Super Mario Bros. worlds, each housing four stages of varying difficulty. These stages are manifested in four different Read-Only Memory (ROM) versions of the game, providing diverse levels of complexity and visual fidelity. In this study we only focus on three of these ROM versions (1).

3.3 State

The Super Mario Bros environment's state space encompasses various elements, including the visual representation of the game, the agent's position, velocity, and the presence of enemies and obstacles. States are observed frames of the game, encapsulating the current snapshot of the environment. Additional information includes the player's score, remaining lives, and the current level.

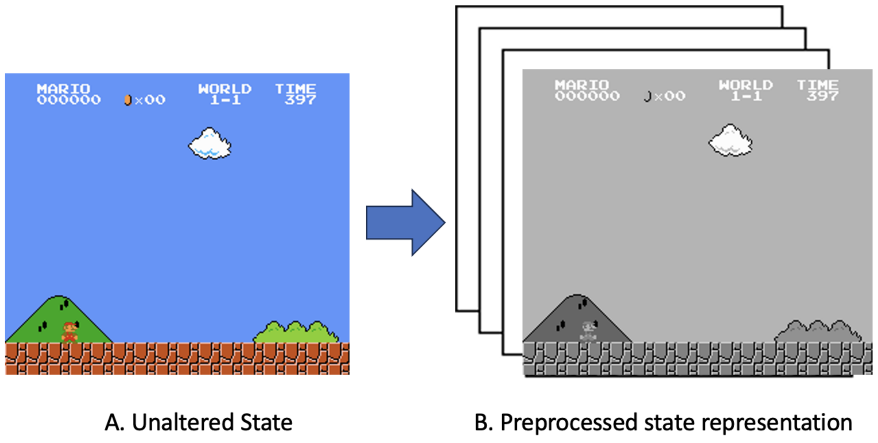


Figure 2: Simple illustration of state preprocessing inputs and outputs.

State Representation Our neural networks are trained on a preprocessed version of the state space, involving the following steps:

Frame Skipping: To reduce redundancy, 4 intermediate frames are skipped between states, as consecutive frames often contain similar information.

Grayscaling: Converting the initial 3 x 240 x 256 RGB window to grayscale reduces dimensionality by $\frac{2}{3}$ (1 x 240 x 256 values) without significantly affecting the agent's decision-making.

Resizing: The image input is resized to an 84 x 84 pixel representation for enhanced processing speed and simplicity.

Frame Stacking: Consecutive frames are stacked to create a consolidated input, allowing the agent to consider temporal aspects in the game easily. The final state representation is a stack of four consecutive, grayscaled, resized frames (4 x 84 x 84).

3.4 Reward System

The reward system is designed to motivate the agent to achieve specific objectives, such as collecting coins, defeating enemies, and completing levels. Positive rewards are assigned for these accomplishments, while negative rewards are given for losing lives or taking damage. The agent's goal is to maximize its cumulative reward throughout an episode. In our implementation, the agent is rewarded for completing the level swiftly while avoiding death.

$$R = Velocity + Clock + Death \tag{1}$$

Velocity: The difference in agent x values between states, promoting movement and penalizing standing still or moving backward.

- $v = x_1 - x_0$
  - x_0 is the x position before the step
  - x_1 is the x position after the step

Clock: The difference in the game clock between frames, penalizing the agent for standing still.

- $c = c_0 - c_1$
  - c_0 is the clock reading before the step
  - c_1 is the clock reading after the step

Death: A penalty for dying in a state, encouraging the agent to avoid death.

- $died = -15$, $otherwise = 0$

3.5 Episode Termination

An episode terminates when Mario completes a level, exhausts all lives, or runs out of time. This episodic structure facilitates the evaluation of the agent's performance across multiple trials.

3.6 Integration with OpenAI Gym

The Super Mario Bros environment seamlessly integrates with the OpenAI Gym toolkit, providing a standardized interface for reinforcement learning experiments. This integration streamlines the comparison and benchmarking of different algorithms. The OpenAI Gym Super Mario Bros environment stands as a comprehensive and challenging testbed for reinforcement learning algorithms, offering a visually rich and dynamic platform that encourages the development of intelligent agents capable of mastering complex gameplay scenarios.

4 Agents

In our exploration of the Super Mario Bros environment, we deploy a range of reinforcement learning agents, each employing distinct algorithms and architectures.

4.1 Double Deep Q-learning Agents

Both of our Double Deep Q-learning agents feature two networks – a target network and an online network – to bolster stability and mitigate overestimation bias. Both utilize an epsilon-greedy algorithm for action selection, starting with an initial epsilon value of 1 and a decay of 0.99. The overall Double Deep Q-learning process unfolds as follows:

1. **Initialization:** Set up the online and target Q-networks with identical architectures (Figures 3 & 4). Configure relevant hyperparameters (Table 1) to define the learning environment.

   a) Experience Replay Buffer: Incorporate the experience replay buffer to store past experiences, alleviating temporal correlations between sequential samples for enhanced learning stability.

3. **Q-Network Update:** Sample a mini-batch of experiences, predict Q-values for actions in the current state using the online Q-Network, and calculate Q-values for actions in the subsequent state using the target Q-network. Update the online Q-network by minimizing the Smooth L1 Loss between the online and target Q-values.

3. **Target Q-Network Update:** Every 10,000 steps, update the target Q-network with weights from the online Q-network. Govern the process with a small constant parameter (tau) for gradual synchronization and training stability.

4. **Action Selection:** Select an action according to the ϵ-greedy policy and Q-value outputs from the online Q-network.

5. **Environment Interaction:** Execute the selected action in the environment and observe the subsequent state and reward.

6. **Experience Replay:** Store the current experiences in the experience replay buffer.

7. **Repeat:** Iteratively repeat steps 2 to 6 for 20,000 training episodes.

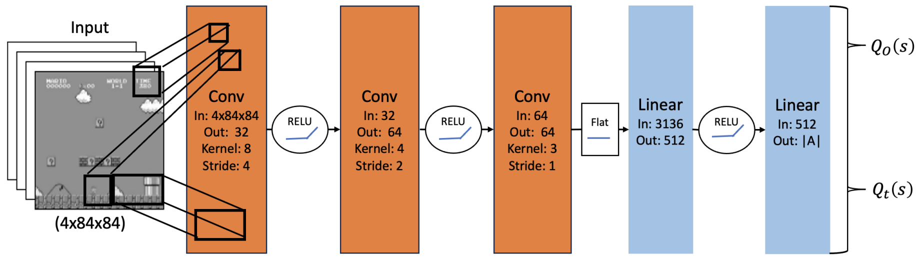


Figure 3: Neural network architecture for baseline DDQN agent.

Baseline DDQN Agent The Baseline DDQN (Double Deep Q-Network) agent serves as our initial reference point. It follows the traditional Q-learning approach outlined in the section header, leveraging the deep neural network architecture shown in Figure 3 to approximate the Q-value function.

Augmented DRDQN Agent To enhance the learning capabilities of the DDQN agent, we introduce the Deep Recurrent Double Q-Network (DRDQN) Agent (as introduced in [12]). This variant incorporates a recurrent LSTM layer after the convolutional torso (Figure 4). The LSTM layer aims to provide the agent with a form of 'active vision,' enabling it to focus on changing areas of the environment. This addition aims to improve the agent's adaptability and capture temporal dependencies in the dynamic Super Mario Bros environment.

4.2 Actor-Critic Agents

Both Actor-critic agents utilize the same underlying neural network architecture (Figure 5), differing in modeling approaches.

PPO Agent The Proximal Policy Optimization (PPO) Agent adopts the actor-critic framework, combining the benefits of policy-based and value-based methods. PPO focuses on stable policy updates by introducing a clipped objective function. The agent uses this approach to iteratively improve its policy while ensuring more reliable and controlled updates.

1. **Initialization:** Initialize the actor and critic neural networks (Figure 5) and configure the relevant hyperparameters (Table 1) to define the learning environment.

2. **Compute Trajectories:** Collect a batch of trajectories from the environment using the current policy.

3. **Compute Advantages:** Compute advantages for each time step in the trajectories using the generalized advantage estimator (GAE).

4. **Update Actor Policy:** Compute the surrogate objective, balancing policy improvement and stability. Perform multiple iterations of optimization using the clipped surrogate objective to avoid large policy updates.

5. **Update Critic Value Function:** Update the value function using the smooth L1 loss between predicted and observed values.

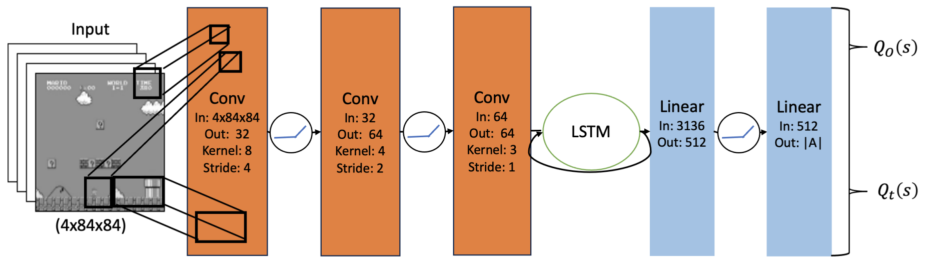


Figure 4: Neural network architecture for Recurrent DDQN agent.

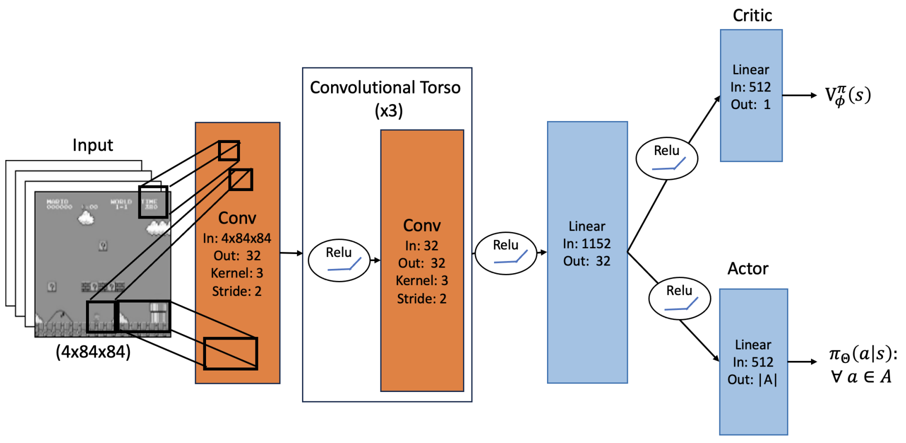


Figure 5: Neural network architecture for Actor-Critic models.

6. **Entropy Regularization:** Regularize the entropy term to encourage model exploration.

7. **Repeat:** Repeat steps 2 to 6 for 20,000 episodes.

A3C Agent The Asynchronous Advantage Actor-Critic (A3C) Agent represents another actor-critic approach with a unique asynchronous training strategy. Multiple agents, each using the actor-critic neural network architecture (Figure 5), run in parallel, each interacting with its own instance of the environment. Based on the experiences of these agents, the algorithm then updates a global model, facilitating faster convergence and efficient exploration.

1. **Initialization:** Initialize the global neural network with actor and critic components.

   a) Create multiple actor threads, each with its own environment instance and a copy of the global network.

2. **Asynchronous Interaction:** Each actor independently interacts with its environment and gathers experiences in the form of (state, action, reward, next state) tuples.

3. **Advantage Calculation:** Compute the advantage function for each (state, action) pair, representing the advantage of taking a specific action in a given state.

4. **Update Global Model:** Compute gradients of the policy and value loss with respect to the global model parameters and update the global model using the gradients, enhancing both the actor and critic components.

5. **Synchronization:** Every 50 steps, update the actor's local model with the parameters of the global model to ensure consistency.

6. **Repeat:** Iterate over steps 2 through 5 asynchronously, with all actors interacting, calculating advantages, and updating the global model.

Table 1: Model Hyperparameters and Training Components

Hyperparameter	**DDQN**	**PPO**	**A3C**
Batch Size	32	16	–
Exploration Rate (ϵ)	1	1	–
ϵ-Decay	$0.9\overline{9}$	$0.9\overline{9}$	–
ϵ-Minimum	0.1	0.1	–
Discount Factor (γ)	0.9	0.9	0.9
Learning Rate (`lr`)	1×10^{-4}	1×10^{-4}	1×10^{-4}
GAE Parameter (τ)	–	1.0	1.0
Entropy Coefficient (β)	–	0.01	0.01
Clipped Surrogate Objective	–	0.2	–
# Epochs	–	10	–
# Parallel Processes	–	8	6
# Sync steps	–	–	50
Seed	42	42	42
Training Components			
Optimizer	AdamW	AdamW	AdamW
Loss Function	SmoothL1Loss	SmoothL1Loss	–

5 Training & Testing

Our experimental setup aims to comprehensively evaluate all baseline and augmented agents in the Super Mario Bros environment. Each experiment runs for 20,000 episodes utilizing the AdamW optimizer for parameter optimization. Table 1 provides the hyperparameters defining the training process for each agent. In this study, we exclusively train the baseline DDQN agent over the Version and Movement curricula.

5.1 Training Environment

Baseline: The RL agent is trained in a specific configuration of the Super Mario Bros environment, set to World 1, Stage 1, with the standard ROM version and simple agent movement. This configuration serves as a foundational training ground for the agent to gradually learn and adapt to the challenges presented in the game.

Naive Version Curriculum: The version curriculum divides training into thirds. In the first third, episodes run over the rectangle ROM; in the middle third, over the pixelated ROM, and in the final third, over the standard ROM. (Figure 1)

Naive Movement Curriculum: The movement curriculum introduces the agent to rightward motions exclusively in the first third of episodes. In the middle third, the agent gains access to simple leftward movements, and in the final third, it obtains the complete suite of complex actions available in the game.

5.2 Testing Environment

The testing environment is designed to assess the agent's generalization and performance. In this scenario, the agent is given ten attempts to tackle each of eight Super Mario Bros. levels, four from each available world. Each agent plays over a standard version of the game with simple agent movement. This environment evaluates the agent's ability to apply learned strategies to different levels and worlds.

6 Experimental Results

We conducted an extensive series of experiments to assess the performance of our Double Deep Q-Network (DDQN) models in the OpenAI Gym Super Mario Bros. environment. To provide context, we compared these DDQN models with our Actor-Critic approaches using Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO). Key metrics for assessment included episode length, reward, loss, and average QValue. Additionally, for the final test, we conducted a distribution analysis covering metrics such as level completion time, achieved reward, distance covered, and the percentage of successfully completed games.

6.1 Training Performance Evaluation

First, we present a detailed analysis of the training performance results for the DDQN-based models (Figure 6). Unfortunately, due to a logging error, we exclude the actor-critic models from this analysis. The baseline DDQN and DRDQN architectures exhibited similar trends in episode lengths, with the DRDQN demonstrating slightly higher rewards. Notably, the movement-curriculum trained DDQN model showed prolonged episodes and diminished rewards early on, constrained to rightward movement. However, it demonstrated significant improvement around episode $\approx$6,600 with the introduction of leftward movements, and another improvement around episode $\approx$13,000 with more complex movements. By the 20,000th episode, the movement-curriculum trained DDQN model displayed the highest rewards. Despite all models indicating a need for further training, the DRDQN uniquely showcased faster learning, with a descending trajectory in its loss curve. Moreover, the DRDQN outperformed other models in average QValue, exhibiting lower entropy in its loss and QValue curves, indicating greater stability.

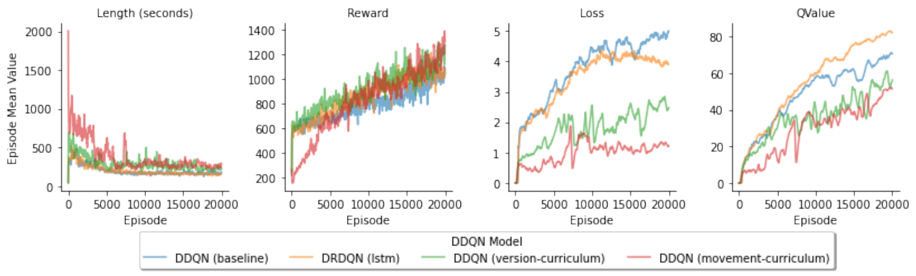


Figure 6: Model performance metrics during training (DDQN model subset).

6.2 Testing Performance Evaluation

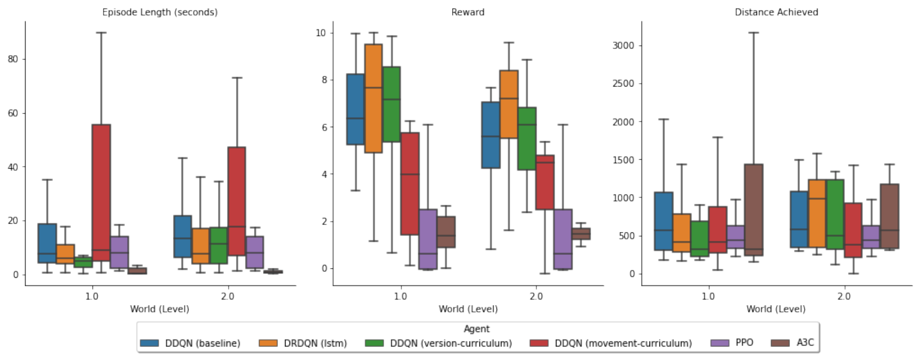


Figure 7: Model performance distributions over all levels in each Mario world.

The most evident result from our evaluation experiments was the completion percentage. The A3C model was the only architecture that completed the first stage of Mario World 1, achieving a 100% completion rate. However, like other models, it failed all subsequent levels. Despite these failures, insights into all the models' performance and generalizability can be gleaned from finer-grained performance measures (Figure 7). The DDQN (movement-curriculum) agent exhibited the longest average episode lengths but among the lowest average rewards and distance covered, suggesting a potentially adverse impact of this curriculum on model performance. In contrast, the version curriculum showed shorter episode lengths and higher average rewards than the baseline

DDQN. The DRDQN model demonstrated the highest average rewards across both worlds and the greatest average distance covered in World 2, indicating superior generalizability. The PPO model displayed the lowest rewards and among the lowest distance covered, pointing to poor generalizability to novel domains. The A3C model, while highly successful on World 1, Stage 1, exhibited low episode lengths and rewards, suggesting overfitting and poor generalization to new environments.

7 Conclusions and Future Work

In conclusion, our investigation into reinforcement learning within the Super Mario Bros environment provided valuable insights into the performance and adaptability of different algorithms and curriculum designs. Experimental results revealed the influence of two training curricula on the learning process, showcasing the strengths and weaknesses of various RL algorithms.

The Double Deep Q-Network (DDQN) models, especially the one trained with a version curriculum, demonstrated improvements in episode lengths and rewards over the baseline model. In contrast, the DDQN model trained with a movement curriculum exhibited inferior performance during evaluations. The incorporation of a recurrent Long Short-Term Memory (LSTM) layer in the DDQN architecture enhanced learning, evidenced by faster convergence and superior stability. Conversely, the Actor-Critic models, specifically Asynchronous Advantage Actor-Critic (A3C) and Proximal Policy Optimization (PPO), showed mixed results. While A3C excelled in completing the first stage of World 1, it struggled to generalize to subsequent levels. PPO, unfortunately, demonstrated poor generalization, achieving lower rewards and distances covered compared to DDQN models (though these results are counterintuitive). Our experiments underscored the importance of thoughtfully designing training curricula, with the version curriculum contributing to improved generalization and the movement curriculum yielding mixed outcomes.

In future work, it would be valuable to explore the impact of incorporating recurrent elements in the neural network architecture across all algorithm classes. Additionally, advanced curriculum learning strategies, such as dynamic and adaptive curricula that adjust based on the agent's learning progress, warrant investigation. The exploration of transfer learning techniques to leverage knowledge gained in one world to accelerate learning in others presents a promising avenue. Fine-tuning hyperparameters and conducting a more extensive hyperparameter search could further optimize model performance.

Overall, our study contributes to the evolving landscape of research on reinforcement learning in video game environments. The insights gained regarding the impact of training curricula and algorithmic choices on agent performance are valuable for advancing the field. As reinforcement learning progresses, addressing challenges in complex environments like Super Mario Bros can lead to advancements with broader implications in real-world applications.

8 Limitations

While our study provides valuable insights into reinforcement learning in the Super Mario Bros environment, several limitations should be considered:

Sensitivity to Hyperparameters The performance of our models is contingent on the choice of hyperparameters. While we conducted a thorough exploration within a defined parameter space, the sensitivity of these models to variations in hyperparameter values will likely impact their generalization and learning capabilities.

Curriculum Design Impact Our experiment utilized two specific curricula—version and movement. While these curricula provided insights, they represent a simplified exploration. The design of curricula is intricate, and the impact of different curriculum structures or variations in the timing of curriculum changes could significantly influence the learning process.

Mistakes in the Existing Study The researchers failed to adequately monitor the training progress of their actor-critic algorithms. This prevented them from including these models in their comparative analysis on training performance.

Algorithm-specific Considerations Each algorithm employed in our study has its own assumptions, strengths, and limitations. A more in-depth analysis of these aspects for each algorithm would provide a more nuanced understanding of their applicability to different scenarios.

Limited Algorithm Exploration Our study focused on Double Deep Q-Networks and Actor-Critic approaches. Other emerging RL algorithms or hybrid models were not explored, potentially limiting the scope of insights into the broader landscape of RL techniques.

Addressing these limitations and further exploring these aspects would enhance the robustness and applicability of this study.

References

[1] Y. Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. volume 60, page 6, 06 2009.

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[3] Yuansong Feng, Suraj Subramanian, Howard Wang, and Steven Guo. Train a mario-playing rl agent, 2023.

[4] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps, 2017.

[5] Christian Kauten. Super Mario Bros for OpenAI Gym. GitHub, 2018.

[6] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016.

[7] Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning, 2017.

[8] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[9] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[12] Felipe Moreno-Vera. Performing deep recurrent double q-learning for atari games, 2019.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[14] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[15] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.