
MaRLo

Playing Super Mario Bros. with various Deep Reinforcement Learning architectures



Sam Donald

ECE 5984: Mathematical Foundation of Reinforcement Learning Algorithms
Virginia Tech
samdonald@vt.edu

Abstract

In this project, multiple Deep Q-Learning frameworks are applied to the 1985 Nintendo game Super Mario Bros as to empirically study their performances. The analyzed networks include Deep Q-Learning (DQN), Double Deep Q-Learning (DDQN), and Dueling Deep Q-Learning (Dueling DDQN), with policies learnt via convolutional neural networks using raw pixel data as input. Over 10000 training episodes, Dueling DDQNs consistently outperformed DDQNs, with DQN models failing to solve the environment. The resulting Dueling DDQN model consistently completes levels 1-1 and 1-4 within 1 second of world record performances. Video clips of these performances from the final stages of training can be found here for levels [1-1](#) and [1-4](#).

1. Introduction

This project initially aimed to implement various Q-learning frameworks within the Super Mario Bros. Open AI Gym environment. However traditional Q-learning methods require many hand-crafted features to interpret the complex 240x256 RGB state space [5, 11], with features needing to be regenerated for different levels. Deep-Q learning methods follow the general structure of standard Q-learning algorithms yet circumvent the need for hand crafted features by utilizing neural networks to approximate value functions. As such Deep-Q learning frameworks are ideal candidates for environments with large state spaces [10]. In this project, Deep Q-learning (DQN)[10], Double Deep Q-learning (DDQN)[1], and Dueling Double Deep Q-learning (Dueling DDQN)[13] networks are applied to Super Mario Bros to analyze their performance. These specific architectures were chosen as they are described as incremental improvements upon each other by Hessel et al. [3].

2. Background

2.1. Related Work

This project is based on Sean Klein’s work [8] along with the MadMario PyTorch Reinforcement tutorial [15], both of which applied DDQNs to Super Mario Bros. The methods outlined within these projects were extended upon to include the Dueling DDQN architecture [13]. The composition of the neural network, along with hyperparameters were based on the breakthrough Atari DQN paper by Mnih et al. [10] and MadMario [15].

2.2 Q-learning background

The goal of Q-learning is to find an optimal Q function, providing the value of an action for a given state. Intuitively this is based on the idea that if an optimal Q function Q^* is known for a given sequence of states, the optimal strategy is that which acts greedily and selects the highest valued action in each state. For a given policy π the true value of action a given state s is defined as followed:

$$Q_\pi(s, a) = E[R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \mid S_0 = s, A_0 = a, \pi] \quad (1)$$

$$Q^*(s, a) = \max_{\pi} Q_\pi(s, a) \quad (2)$$

Where the discount factor $\gamma \in [0,1]$ is used to balance between immediate and delayed rewards, and R_n is the reward at step n .

Q-learning can be implemented to iteratively converge to the optimal Q function [7]. However, this requires knowledge of all state and action values, which is computationally infeasible for large problems, and as such a parameterized version is learnt. Typically, linear function approximators are used, but this can be expanded upon to utilize neural networks. The update procedure is then as follows, with α being a scalar step size, and θ the parameterized value function:

$$\theta_{t+1} = \theta_t + \alpha [Target - Q_{local}] \nabla_{\theta_t} Q_{local} \quad (3)$$

$$Q_{local} = Q(S_t, a_t; \theta_t) \quad (4)$$

$$Target = R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a'; \theta_t). \quad (5)$$

To ensure exploration, actions are chosen epsilon greedily, with epsilon chance of taking a random action as opposed to the action associated with the maximum Q value. All experiments within this project utilized the following epsilon decay function:

$$\varepsilon = \max(0.9995^{episode}, 0.02) \quad (6)$$

2.1 Super Mario Bros.

The goal of Super Mario Bros. is to maneuver player character Mario to the end of the level, while avoiding various obstacles and enemies. The environment provides a score, which is calculated through a combination of acquiring coins, defeating enemies, and completing the levels with time remaining. Within this project, the gym-super-mario-bros library for OpenAI Gym [14] was utilized, in conjunction with a NES emulator. The inbuilt reward function was utilized, which is based on a combination of horizontal movement and time to prevent stagnation. A detailed description of this reward function available on the gym-super-mario-bros library reference page [14].

2.1.1 Action space

As the original version of the Super Mario Bros. was designed for the Nintendo Entertainment System (NES), inputs are provided via a four directional joystick (Up, Down, Left, Right) along with two buttons (A, B). The horizontal directions are used to move Mario left and right, the A button performs a vertical jump, and the B button causes Mario to sprint. The Up and Down commands are used to perform infrequent actions such as climbing ladders or moving into pipes. While the action space is small, comprising 11 unique combinations, Mario’s velocity can be constantly modified via the left and right commands along with jumping to various heights based on longer jump commands. Through these combinations, exceptionally precise actions are possible.

Movements in the gym-super-mario-bros environment movements are categorized into three distinct categories and are listed within table 1. This project utilizes both Right Only and Simple Movement actions spaces, preventing Mario from taking shortcuts which are accessed by moving down into certain pipes.

	Right Only	Simple Movement	Complex Movement
No action	Yes	Yes	Yes
Right	Yes	Yes	Yes
Right + A	Yes	Yes	Yes
Right + B	Yes	Yes	Yes
Right + A + B	Yes	Yes	Yes
A	-	Yes	Yes
Left	-	Yes	Yes
Left + A	-	-	Yes
Left + B	-	-	Yes
Left + A + B	-	-	Yes
Down	-	-	Yes
Up	-	-	Yes

Table 1: List of available actions for different action sets in the gym-super-mario-bros environment.

2.1.3 Environments and Enemies

Super Mario Bros. levels are divided into four unique categories: overworld, underworld, underwater and castle, each with unique defining obstacles and enemies. Overworld level 1-1 and castle level 1-4 are used as training environments within this paper, with full levels displayed within figure 1. Level 1-1 is noted as being one of the most iconic video game levels of all time.

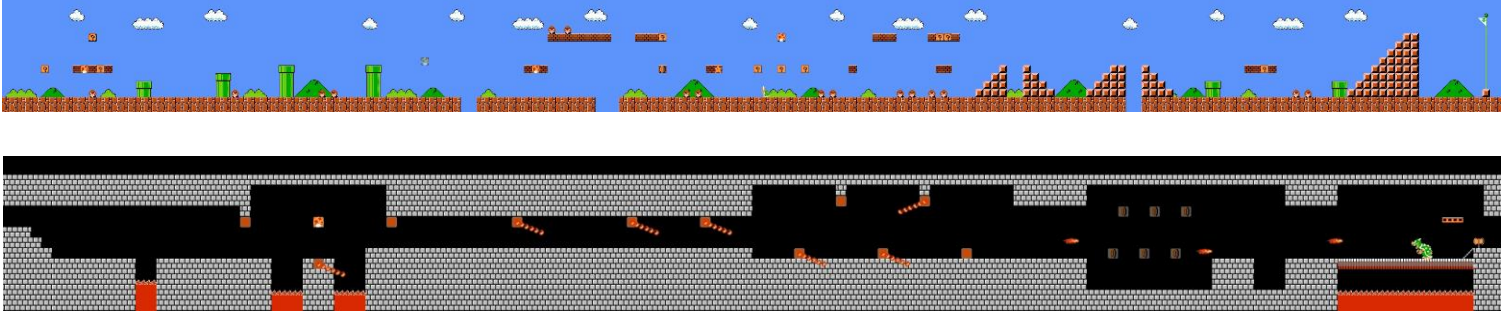


Figure 1: The entirety of levels 1-1 and 1-4 (top and bottom respectively).

Throughout these levels, Mario is presented with a multitude obstacles and enemies which must be avoided, summarized by tables 2 and 3. Of note, typical human players will acquire various powerups throughout levels by destroying specific blocks, these powerups increase Mario's size and give him one free contact with an enemy before a life is lost. These powerups are not described as they were not acquired during training instances. As such, contacting enemies or a Fire-Bar will result in Mario's death, excluding jumping directly on top of certain enemies which is referred to as a stomp.




Sprite	Name	Description	Level
	Little Goomba	A mushroom that walks back and forth. Can be stomped on.	1-1
	Koopa Trooper	A turtle that walks back and forth. Can be stomped on to generate a shell, which can be kicked to hit other enemies.	1-1
	Fake Bowser	Boss of world 4. Shoots fire balls horizontally and jumps. Is unable to be stomped.	1-4

Table 2: Enemies present in levels 1-1 and 1-4






Sprite	Name	Description	Level
	Pit	Environmental trap resulting in death.	1-1 1-4
	Pipe	Vertical obstacle that must be jumped over.	1-1
	Blocks	Blocks of various kinds that prevent Mario's movement, with some being invisible in world 4.	1-1 1-4
	Stairs	Stacked blocks that must be sequentially climbed.	1-1
	Fire-Bar	Rotating bars of fire resulting in death, can rotate both clockwise and anticlockwise.	1-4

Table 3: Environmental obstacles present in levels 1-1 and 1-4

4. Methodology

4.1 Preprocessing

By default, a single observation of a game frame is a 240x256 RGB image. As processing this amount of data is computationally expensive [13], preprocessing is required to reduce input dimensionality. Raw RGB frames are first converted to a normalized greyscale range, before being cropped to 182x256 to remove redundant information such as the score and repeated textures below ground. This is then rescaled to the final 84x84 resolution, which is square such that 2D convolutions can be applied. Figure 2 shows example outputs from this procedure.

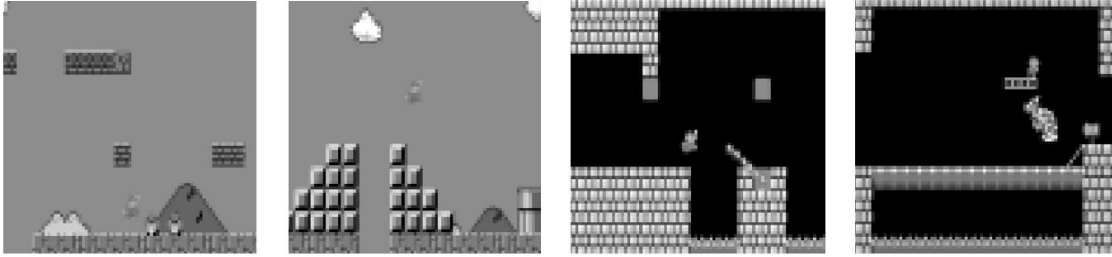


Figure 2: Processed frames, with the left two being from level 1-1 and the right two from 1-4. The rightmost image includes the final boss, Bowser.

These steps are applied to 4 sequential frames and then stacked together to incorporate a sense of motion, with figure 3 visualizing the complete process.

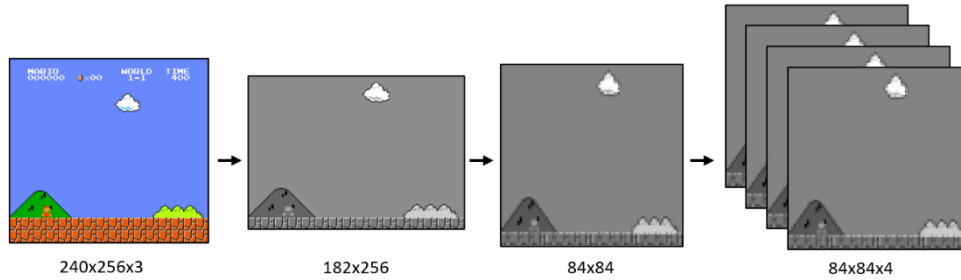


Figure 3: Preprocessing workflow, including greyscale normalization, cropping, rescaling, and frame stacking.

4.2 Replay Buffer

Processed frames are stored in a replay buffer of size 30,000, with new frames pushing out the oldest. All architectures implemented within this paper uniformly sample from this buffer when completing the update step of Q-learning. Q-learning updates are incremental and do not converge quickly, so multiple passes with the same data is beneficial when there is low variance in immediate outcomes between sequential states. Through implementing this method, better convergence behavior can be observed as data better conforms to the i.i.d assumptions made in Q-learning convergence proofs. This process is outlined in detail by Mihn et al. [10].

4.2 Model Architectures

4.2.1 Deep Q-Learning (DQN)

Deep Q networks (DQNs) are characterized by having a multi-layered neural network acting as a function approximator between given states and output action values [10]. The target function described by Equation 5 is redefined as:

$$Target_{DQN} = R_{t+1} + \gamma \max_a Q_{local}(S_{t+1}, a). \quad (7)$$

With the loss function over which gradient descent is performed defined as:

$$Loss = \left(Target_{DQN} - Q_{local}(S_t, a_t) \right)^2 \quad (8)$$

The DQN implemented comprises of the following layers, all of which incorporate rectified linear unit (ReLU) activation functions [9]. This configuration is visualized in figure 4:

- Input: 84×84×4 preprocessed frames.
- Hidden layer 1: Convolves 32 8×8 filters with stride 4.
- Hidden layer 2: Convolves 64 4×4 filters with stride 2.
- Hidden layer 3: Convolves 64 3×3 filters with stride 2.
- Hidden layer 4: Fully connected 512 ReLUs.
- Output layer: Fully connected linear layer, based on action space size (5 or 7).

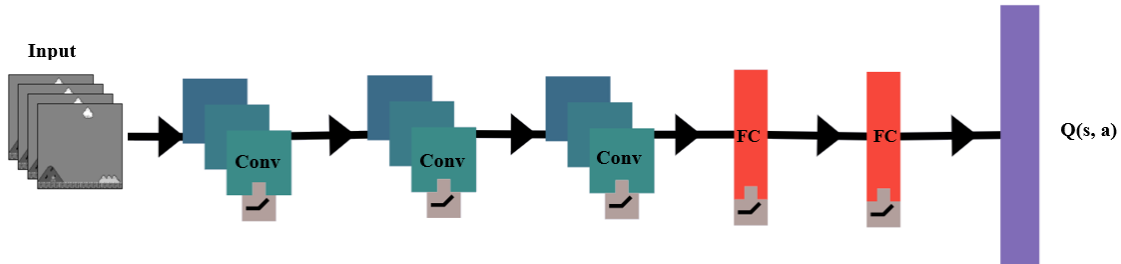


Figure 4: Visualization of the DQN neural network structure.

4.2.2 Double Deep Q-Learning (DDQN)

The primary issue with DQN is related to the max operation of Equation 7, which results in the same set of values being used to both select and evaluate actions, ultimately increases the likelihood of overestimation [1]. Double Deep Q networks (DDQNs) aim to address this issue by separating selection and evaluation. This process mimics that of Double Q learning in the tabular setting, where two value functions are learned simultaneously by updating either function at random. During the update step, one network is used to determine the greedy policy, while the other is using to determine its value.

Mimicking this approach, DDQN incorporates two networks in the learning process, Q_{target} and Q_{local} . While both are parametrized by θ , Q_{target} has its parameter values cloned from the Q_{local} network every τ steps. Within this project a value of 5000 steps was used. The target of the DDQN is defined as followed, with a loss function identical to Equation 8:

$$Target_{DDQN} = R_{t+1} + \gamma Q_{online} \left(S_{t+1}, \underset{a'}{\operatorname{argmax}} Q_{target} (S_{t+1}, a') \right). \quad (9)$$

Instead of directly copying Q_{local} into Q_{target} , Polyak averaging can be implemented to incrementally update the weights of Q_{target} in accordance with Equation 10, where β is typically set to 0.01 [1].

$$\theta_{target} \leftarrow \beta \theta_{local} + (1 - \beta) \theta_{target} \quad (10)$$

By applying these methods, it has been shown that overestimations present in DQNs are eliminated and overall performance improves [1, 3]. The structure of the neural network used for both the Q_{target} and Q_{local} networks during DDQN experiments is identical to the implemented in the DQN.

4.2.3 Dueling Double Deep Q-Learning (Dueling DDQN)

The idea behind dueling networks is that for many states it is unnecessary to estimate the value of every action choice, as some evaluations are only pertinent in specific situations. In the context of Super Mario Bros., evaluating different actions while in a wide-open area free of enemies is not necessary, whereas it is exceptionally important when an enemy is approaching, or a pit is nearby.

The initial layers of the dueling neural network are identical to the aforementioned DQN and DDQN, however the fully connected layers are instead replaced with two branches providing estimates of the value function $V(s)$ and the advantage function $A(s, a)$. These two branches are then combined to output a set of Q values for each possible action. This configuration is visualized within figure 5.

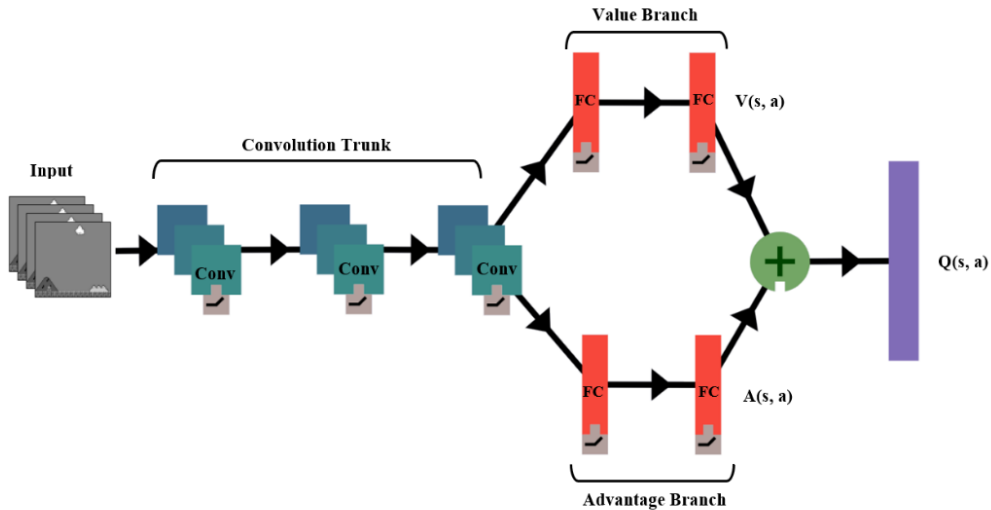


Figure 5: Visualization of the Dueling DDQN neural network structure.

Layer configurations are the same as per the DQN and DDQN, however the final fully connected layer from the value branch is instead mapped to a singular scalar value.

While it appears intuitive to combine $A(s, a)$ and $V(s)$ via Equation 11, this results in an unidentifiable value of $Q(s, a)$, such that $A(s, a)$ and $V(s)$ cannot be recovered uniquely. Wang et al notes that this results in poor practical performances [13]. Instead, Equation 12 is used, forcing the Q value of the maximized action to be equal to V and addresses the issue of identifiability.

$$Q(s, a) = V(s) + A(s, a) \quad (11)$$

$$Q(s, a) = V(s) + A(s, a) - \max_{a'} A(s, a) \quad (12)$$

As the dueling architecture shares the same input-output interface with the DDQN and DQN, the same loss functions and training process can be applied.

5. Results

Training of networks was completed on multiple RTX 6000s via Lambda Cloud servers. Each model was trained for 10000 episodes for an average of 1.5 million steps. The singular addition that was made to the described models was a stuck flag, which terminated the episode should mario not progress forward for 50 consecutive frames. Descriptions of experiments conducted along with training times are listed in table 4.

Experiment ID	Model	Level	Movements Set	Additions	Training Time [hours]
DQN_1-1_right	DQN	1-1	RIGHT_ONLY	-	9.9
DQN_1-1_right_stop	DQN	1-1	RIGHT_ONLY	Stuck flag	6.7
DDQN_1-1_right	DDQN	1-1	RIGHT_ONLY	-	8.1
DDQN_1-1_right_stop	DDQN	1-1	RIGHT_ONLY	Stuck flag	7.4
DuelDQN_1-1_right	Duel DDQN	1-1	RIGHT_ONLY	-	15.6
DDQN_1-4_right	DDQN	1-4	RIGHT_ONLY	-	9.9
DDQN_1-4_simple	DDQN	1-4	SIMPLE_MOVEMENT	-	10.1
DuelDQN_1-4_right	Duel DDQN	1-4	RIGHT_ONLY	-	11.0
DuelDQN_1-4_simple	Duel DDQN	1-4	SIMPLE_MOVEMENT	-	9.2

Table 4: Experiment descriptions

5.2 World 1-1

All DQN agents experienced negligible learning over 10000 episodes and never successfully completed the level. DDQN agents however were able to reach consistent scores above 2000 after 6900 episodes, at which they began to plateau. The Dueling DDQN agent consistently outperformed all others, able to reach scores over 2000 after 6100 episodes, plateauing at approximately 2200, along with a final completion rate of 60.3%. It is noted that this completion rate was associated with a 2% chance of random actions due to Equation 6. Unfortunately logging of level completion statistics was not enabled for any agent besides Dueling DDQN, yet it can be inferred that completion rates were lower based on average scores. Results from training are presented in figure 6.

The DDQN agent with the included stop flag was consistently outperformed by the baseline DDQN agent throughout all stages of training. However, training was completed 31.2% faster and as such could be deemed useful when optimizing hyperparameters.

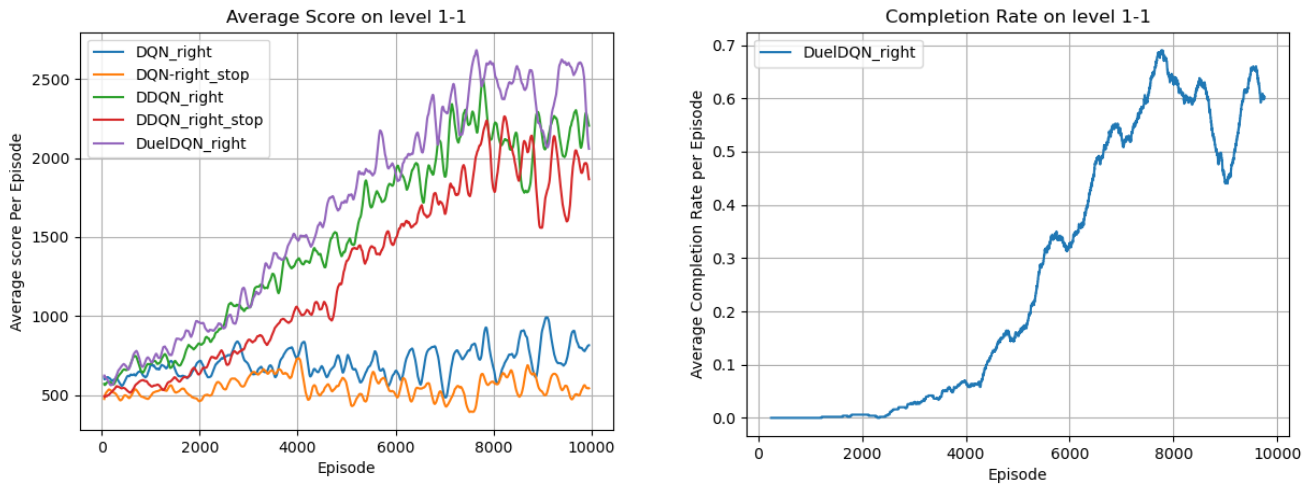


Figure 6: Left plot showing average reward per episode of level 1-1 during training, right plot displaying completion rate per episode for the Dueling DDQN agent

5.3 World 4-1

All agents within world 1-4 were able to successfully complete the level. The Dueling DDQN agent using right only actions performed marginally better than any other agent, attaining a final average score of 1810 and a completion rate of 50.1%. Interestingly, the DDQN agents performed nearly identically regardless their action spaces, implying that the addition of left movement was not useful within this level. The Dueling DDQN agent with simple action set experienced significant oscillations in performance towards the conclusion of training, resulting in a final performance on par with that of DDQN agents.

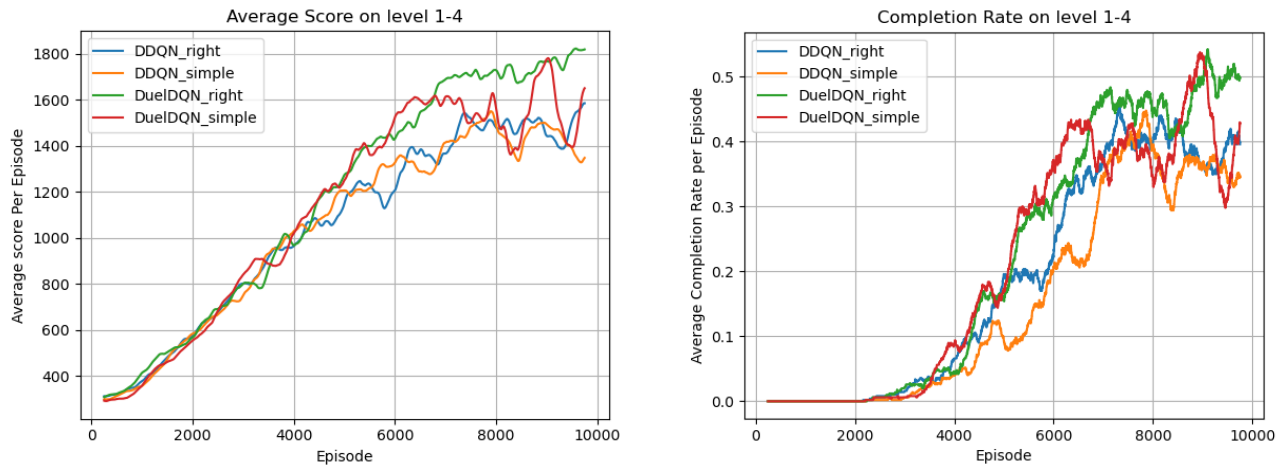


Figure 7: Left plot showing average reward per episode of level 1-4 during training, right plot displaying completion rate per episode.

5.4. Epsilon Decay

On manual inspection of the trained agents, most incomplete runs were a direct result of random actions chosen in accordance with Equation 6. These actions primarily caused mario to jump up at times when a Fire-Bar or Goomba was directly overhead. Modifying Equation 6 to instead decay to zero is hypothesized to result in perfect completion rates as the Super Mario Bros. environment is deterministic.

5.5. World Clear Times

MaRLo was benchmarked against multiple levels of human performance, with the time of MaRLo determined by taking the best of 10 runs from the Dueling DDQN agent with right only action set. MaRLo was able to complete both levels 1-1 and 1-4 within one second of world record tier times, while significantly surpassing the scores attained by typical casual players as listed in table 5.

Level	MaRLo	World Record	Personal Best	Average Human [5]
1-1	53	52*	57	64
1-4	38	37*	40	57

Table 5: Comparison of level completion times for level 1-1 and 1-4. World record tier scores were attained from analyzing video recording from records without shortcuts [16]. Personal best times were determined from 20 minutes of play.

Links to YouTube recordings displaying the final stages of training of the Dueling DDQN agent with right only actions for world [1-1](#) and [1-4](#) can be found here, with 3/4 and 7/9 runs respectively completed at near world record pace.

6. Conclusion

Within this project, multiple Deep-Q learning frameworks were trained to complete levels 1-1 and 1-4 of Super Mario Bros. Of the configurations tested, it was identified that the baseline DQN architecture was insufficient to solve level 1-1, while the Dueling DDQN architecture with right only action set outperformed all others in both levels. These empirical results confirm the incremental improvements outlined by Hessel et al [3].

8. Future Work

Many potential improvements can be applied to this project, as most architectures in the area of deep reinforcement learning are iterative. Hessel et al. [3] describes these improvements in detail via their Rainbow algorithm. Of the techniques mentioned, a prioritized replay buffer is expected to greatly improve training times by allowing for targeted training on rarely observed states. Additional techniques include noisy DQNs [4], and random network distillation [12] to improve the exploration strategy over the elision greedy approach implemented. Finally, hyperparameter tuning is expected to significantly improve performance and stability of the agents yet was not conducted due to the exceptionally long training times.

9. References

- [1] Hado van Hasselt, Arthur Guez, & David Silver. (2015). Deep Reinforcement Learning with Double Q-learning.
- [2] Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems. Curran Associates, Inc.
- [3] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, & David Silver. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning.
- [4] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, & Shane Legg. (2019). Noisy Networks for Exploration.
- [5] Nan Zhang & Zixing Song (2019). Super Reinforcement Bros
- [6] Qingfeng Lan, Yangchen Pan, Alona Fyshe, & Martha White. (2021). Maxmin Q-learning: Controlling the Estimation Bias of Q-learning.
- [7] Richard Sutton and Andrew Barto. Reinforcement Learning: An Introduction. MIT Press, 1998.
- [8] Sean Klein (2016). Deep Q-Learning to Play Mario
- [9] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pages 807–814, 2010.
- [10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, & Martin Riedmiller. (2013). Playing Atari with Deep Reinforcement Learning.
- [11] Yizhen Liao, Kun Yi, Zhe Yhan (2012). Reinforcement Learning to Play Mario
- [12] Yuri Burda, Harrison Edwards, Amos Storkey, & Oleg Klimov. (2018). Exploration by Random Network Distillation.
- [13] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, & Nando de Freitas. (2016). Dueling Network Architectures for Deep Reinforcement Learning.
- [14] <https://pypi.org/project/gym-super-mario-bros/>
- [15] YuansongFeng, Mad Mario, <https://github.com/YuansongFeng/MadMario>
- [16] <https://www.speedrun.com/smb1#Warpless>