Reinforcement Learning Project

Anonymous Authors

***Abstract*—This research project delves into the training of an intelligent agent within the dynamic environment of Super Mario, employing various methodologies to attain optimal performance. Three distinct training strategies are explored: playing using random policy, playing using trained Deep Q-Networks (DQN), and playing using trained Dueling DQN. Through meticulous experimentation and analysis, we assess the comparative efficacy of these methodologies. Our findings suggest that Dueling DQN, which separates the Q-function into value and advantage components, shows promise in enhancing agent performance when compared to conventional DQN methods. However, further exploration is deemed necessary to solidify this assertion. Specifically, a study about the limitations of the Dueling DQN is crucial to validate the robustness and applicability of these observations.**

I. Introduction

Reinforcement Learning (RL) offers a powerful framework for training intelligent agents to make sequential decisions in dynamic environments. In this study, we focus specifically on the application of RL to train an agent within the iconic Super Mario environment. Our exploration delves into various training methodologies, with a keen focus on the comparison between standard Deep Q-Networks (DQN) and the innovative Dueling DQN approach.

In this work, we will be concentrating on the training of an RL agent to navigate and complete levels within the Super Mario environment. This involves the agent observing the state of the game, selecting actions, and receiving feedback in the form of rewards or penalties.

The study of RL in the context of Super Mario is intriguing due to its relevance to real-world scenarios where autonomous agents must navigate complex environments with limited information. Understanding how RL techniques can be applied to such environments can lead to advancements in fields such as robotics, autonomous driving, and game AI.

Thus, one of the primary challenges that we can face in training an RL agent for Super Mario is the high-dimensional and stochastic nature of the environment. The agent must learn to navigate through a variety of obstacles, enemies, and terrain while optimizing for long-term rewards. This environment obliges us to go for a deep learning method to train the agent.

Indeed, prior research in RL has explored various techniques for training agents in video game environments, including Super Mario. Approaches such as DQN have shown promise in learning effective policies, but there remains room for improvement, particularly in terms of sample efficiency and performance. In this report, we will be trying to dive into possible improvements we canmake to enhance the existed techniques in RL.

Our approach involves three distinct phases of training: initial training with a random policy, subsequent training using DQN, and a final training using Dueling DQN. We employed convolutional neural networks combined with fully connected networks to process the game state and estimate Q-values, and we utilized experience replay to improve sample efficiency.

Our experiments reveal that Dueling DQN outperforms standard DQN in terms of agent performance, demonstrating the effectiveness of partitioning the Q-function into distinct value and advantage components. This implies that focusing on the relative importance of actions (which is the Advantage Function) can lead to more efficient learning and better decision-making in complex environments.

While our results are promising, there are limitations to consider, including the need for further analysis of hyperparameters and the scalability of our approach to more complex environments. Future research could explore more the limitations of the Dueling DQN by changing the version of the Super Mario environment. (In this work we used the version SuperMarioBros-v0)

You can find here our work: **Colab notebook**.

II. Background

A. Generality

For our project, we needed some basics about the RL framework. In many RL scenarios, an agent observes a series of states $s_t \in S$ at each time step t [1]. The agent selects an action from a discrete set $a_t \in A$ and observes a reward signal r_t from the environment.

The primary objective of the agent is to maximize the expected discounted return, defined as:

$$R_t = \sum_{\tau=0}^{\infty} \gamma^{\tau} r_{\tau}.$$

$\gamma \in [0, 1]$ serves as a discount factor, influencing the importance of immediate versus future rewards.

Given an agent following a stochastic policy π, the values associated with state-action pairs (s, a) and states s are expressed as follows:

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

and

$$V^{\pi}(s) = \mathbb{E}_{a \sim \pi(s)}[Q^{\pi}(s, a)].$$

Furthermore, Researchers defined an important quantity, the advantage function, which relates the value and Q functions [3]:

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s).$$

The value function V evaluates the desirability of a state s, while the Q-function assesses the value of selecting a specific action in that state. The advantage function, by subtracting the state's value from the Q-function, provides a measure of the relative importance of each action.

B. DQN

Deep Q-networks (DQN) operates in high-dimensional spaces to approximate Q-value functions [4]. A deep Q-network is represented as $Q(s, a; \theta)$ with parameters θ. To train this network, a sequence of loss functions is optimized at iteration i, defined as:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s_0}\left[\left(y_i^{DQN} - Q(s, a; \theta_i)\right)^2\right],$$

where $y_i^{DQN} = r + \gamma \max_{a'} Q(s_0, a'; \theta^-)$ and θ^- represents the parameters of a fixed and separate target network.

This approach is model-free, as the states and rewards are produced by the environment. It is also off-policy because the states and rewards are obtained with a behavior policy (such as epsilon-greedy). For our project, we will be using convolutional layers followed by fully connected layers to create the Deep Q-Network.

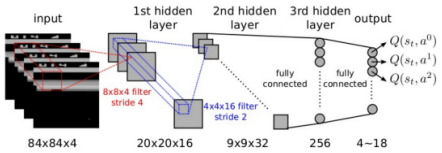


Fig. 1: DQN Architecture [2]

C. Dueling DQN

In the Dueling DQN, Researchers separated the Q-function into two components: one that estimates the Value Function and the other that estimates the Advantage Funtion [3]. They employed two sequences of fully connected layers after the convolutional layers then combined them to generate a single output Q-function.

In the Dueling Network, one sequence of fully connected layers outputs a scalar $V(s; \theta, \beta)$, while the other outputs an $|A|$-dimensional vector $A(s, a; \theta, \alpha)$, where θ represents the parameters of the convolutional layers and α and β represent the parameters of the fully connected layers.

An initial attempt to construct the aggregating module would be to use the following equation:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

However, Researchers suggested that this approach is flawed as it leads to identifiability issues.[3] To address this, they introduced a modified module:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \max_{a_0 \in |A|} A(s, a_0; \theta, \alpha)\right)$$

Training of the Dueling Network, similarly to standard Q networks, involves back-propagation, automatically computing $V(s; \theta, \beta)$ and $A(s, a; \theta, \alpha)$ without additional supervision or modifications.

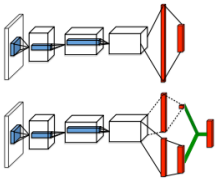


Fig. 2: Dueling DQN Architecture

III. Methodology/Approach

A. Environment

To implement our project, we started by setting the Super Mario Environment using the package gym-super-mario-bros (You can find documentation about this environment here [5]). The agent in this environment is Mario. He had 7 possible actions (These actions were defined basing on the buttons on Nintendo devices: button A and B):

- 'NOOP': This action means the agent does nothing. It's equivalent to not pressing any buttons on the controller.
- 'right': This action makes Mario move to the right.
- 'right A': This action makes Mario to the right while jumping.
- 'right B': This action makes Mario moves to the right while shooting fireballs.
- 'right A B': This action makes Mario moves to the right while jumping, and shooting fireballs.
- 'A': This action makes Mario jump.
- 'left': This action makes Mario move to the left.

The reward function is designed to guide the agent towards the objective of advancing as far to the right as possible, as quickly as possible, while avoiding death. This reward is then clipped into the range (-15, 15). The death penalty is equal to -15. The main component that controls the reward is velocity.

If velocity is positive, the reward is positive. If it's negative, the reward is negative. The more we have a higher value of velocity, the more there is reward (a high value of velocity means that the agent will advance the game clock and make progress by moving to the right)

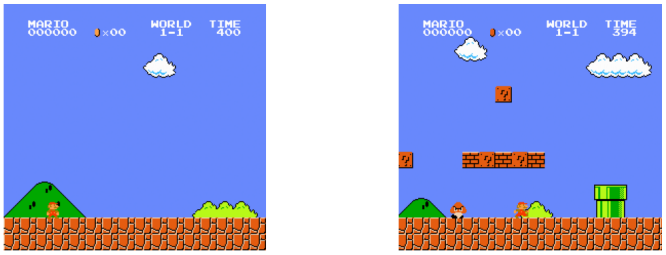


Fig. 3: Super Mario Environment

B. Random Policy

Then, we started testing our agent on a random policy with a maximum number of steps equal to 500. Because we are doing this randomly, we decided to keep only 3 basic actions ['right', 'left', 'A']. The training was done by iterating through the number of steps and using this instruction:

```
observation, reward, done, info=env.step(
    valid_actions[np.random.randint(3)]
)
```

we extracted the reward and the info['x_pos'] which informs about the position of the agent.

C. DQN Approach

Then we started defining our networks to train our agent in an efficient way. As the goal of the project is to implement the Dueling DQN, we started by implementing a simple DQN. To do that, we defined a class **QNetworkCNN** in which we defined the architecture.

As an architecture, we chose to use 5 convolutional layers with a padding of 1 and kernel 3×3 to maintain the spatial dimensions of the input. The number of output channels increases progressively from 4 to 16 while passing through the network, allowing the network to capture increasingly complex features.

After each convolutional layer, we applied a ReLU activation function then a max-pooling of 2×2 to reduce the spatial dimensions. Then, we flatten the output to be fed into the fully connected layers. We decided to use two fully connected layers with ReLU activation applied to the hidden layer and no activation applied to the output layer as we want to get the Q-values for each action as a result.

D. Dueling DQN Approach

To implement this architecture, we defined a class **QNetworkDuellingCNN** in which we defined the convolutional layers and the MLP used.

For the convolutional part, we used the same convolutional layers as the DQN. The main difference remains in the fully connected layers as the dueling architecture separates the representation of the value and advantage functions, allowing the network to learn which states are valuable and which actions are advantageous independently.

For the MLP part, two sets (for the Value Function and Advantage Function) of fully connected layers were used, one for the value function and one for the advantage function. For each set, we used two layers. For the Value Function set, the last layer was outputing a single value with dimension 1 as the goal is to output the state's value. For the Advantage Function set, the last layer was outputing a result with the dimension action_size as the goal is to produce Q-values for each action.

For the forward pass, we made the same thing as with the DQN. Having two fully connected layers, we passed the flattened output to it, to get the Value Function and the Advantage Function (for this function, we adjusted it with mean advantage to stabilize the learning process and improve the convergence of the dueling network).

E. Memory buffer

The Replay Buffer technique came as a solution to the fact that the input data can be highly correlated (as samples are not iid) which can cause overfitting, forgetting, bias the network to learn just a replay.[2] In our project, we defined a class ReplayBuffer which goal is to store experiences (state, action, reward, next state, done) observed by the agent during interactions with the environment.

In the class we created, we defined methods such as "add" method, "sample" method, "update_error" method. The "sample" method randomly samples a mini-batch of experiences from the buffer helping break the temporal correlations present in consecutive experiences and preventing the agent from focusing too much on recent experiences.

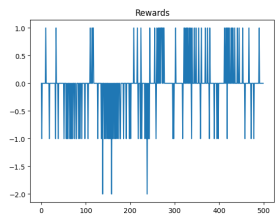
F. Training

To initialize the agent, we decided to intialize it using a two networks (local and target network) and to use a parameter that we called ddqn to choose if we will be computing the loss in a DQN way or in a Dueling DQN way.

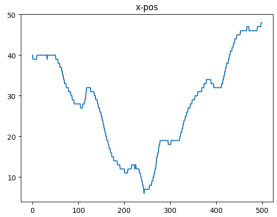
For the hyperparameters, we chose to train with 100 episodes, a maximum number of steps equal to 500, a discount factor of 0.99 and a learning rate of 0.0005.

IV. Results and Discussion

We started our experiments by testing the agent on Random Policy. The agent got -17 as a sum of rewards. This is due to the fact that the agent is going right and left randomly. When the agent goes left, he gets negative reward and when he goes right, he gets positive rewards. In the position figure below, we can see that at certain number of steps, the agent went to the left decreasing its position and increasing in the number of negative rewards (between the steps 100 and 200, there is no positive reward).



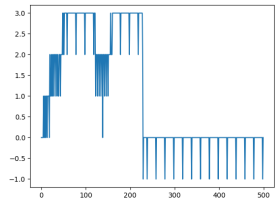
(a) Rewards gained by the agent



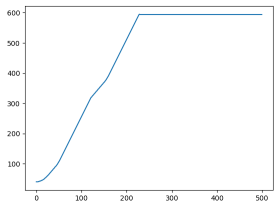
(b) Position of the agent in the game

Fig. 4: Rewards and position of the agent playing with a random policy

Then, we started experimenting with the DQN. We trained our DQN on 100 episodes making it capable to choose actions to maximize the gain of the agent. The agent reached, when playing, a total of rewards equal to 529. This number is way to far from the result we got using random policy, which is totally normal as the role of the Neural Network is to find the best step to adopt. However, at a certain number of steps, we can see in the rewards figure, that the agent started cumulating negative rewards without finding a solution to change the situation. This can also be seen in the position figure, as the agent got stuck at a certain position which means that it couldn't overcome a certain obstacle.



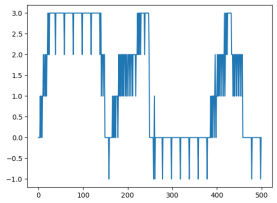
(a) Rewards gained by the agent



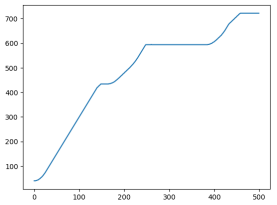
(b) Position of the agent in the game

Fig. 5: Rewards and position of the agent trained using a DQN

Finally, To further advance our study, we experimented on the Dueling DQN. We trained a Dueling DQN architecture to give the actions that the agent needs to follow. When playing the game, the agent gained a sum of rewards equal to 657. This result is better than the one obtained using a simple DQN. In the rewards figure, we can see that a certain number of steps, the agent was in the same situation as if we were using a simple DQN. However, we can notice that the agent found a solution to overcome the obstacle he found (which can be concluded from the posX figure). This shows the power of partitioning the Q-function into a Value Function and an advantage Function.



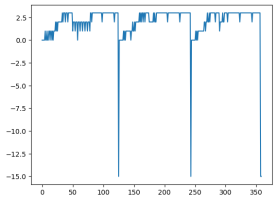
(a) Rewards gained by the agent



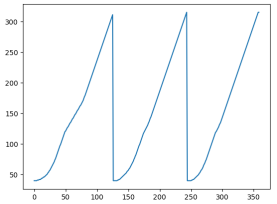
(b) Position of the agent in the game

Fig. 6: Rewards and position of the agent trained using a Dueling DQN

At this level, it's obvious to say that the Dueling DQN presents a better performance than the simple DQN, however when running the code more times, we got some "funny" results that can be seen as a limitation for the Dueling DQN. In the rewards figure below, we can notice that the agent died three times (as he got three rewards equal to -15) and all the other rewards were positive. The same thing can be noticed in the posX figure,as the game is reinitialized each time the agent die. When visualizing the animation of the agent playing, we found that the agent chose to die. In order to maximize the rewards, the agent chose to die each time hitting the first monster he finds, and get positive rewards from walking (to the right) from the beginning to the position of the monster. This strategy opted by the Dueling DQN, shows that the network found another way to maximize the gain through the game, which may be seen as a limitation of this architecture. As a next step, we will be diving more in this problem and searching for solutions (One solution can be to penalize more the death, but increasing the number os steps will make this penalization insignificant)



(a) Rewards gained by the agent



(b) Position of the agent in the game

Fig. 7: Rewards and position of the agent trained using a Dueling DQN

V. Conclusions

In this study, we delved into the challenging environment of Super Mario to train an intelligent agent. Through meticulous experimentation, we compared standard Deep Q-Networks (DQN) with the innovative Dueling DQN approach. Our findings highlight the efficacy of Dueling DQN in enhancing agent performance, emphasizing the importance of exploring novel architectures and methodologies to advance reinforcement learning techniques. However, our research has limitations. Further exploration is necessary to validate the robustness and applicability of our observations across diverse environments.

Additionally, an investigation of the limitations of the Dueling DQN needs to be done.

In conclusion, our research underscores the potential of Dueling DQN to push the boundaries of RL capabilities. By continuing to innovate and refine methodologies, we can unlock new opportunities for intelligent agent development and drive advancements in AI-driven systems.

REFERENCES

[1] Sutton and Barto. Reinforcement Learning, *MIT Press*, 2020.

[2] Read. Lecture VI - Reinforcement Learning III. In *INF581 Advanced Machine Learning and Autonomous Agents*, 2024.

[3] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, N. Freitas *Dueling Network Architectures for Deep Reinforcement Learning*, **arXiv:1511.06581v3 [cs.LG]**., April 2016.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller *Playing Atari with Deep Reinforcement Learning*, **arXiv:1312.5602v1 [cs.LG]**., December 2013.

[5] Visited *gym-super-mario-bros 7.4.0*, **gym-super-mario-bros**.

APPENDIX

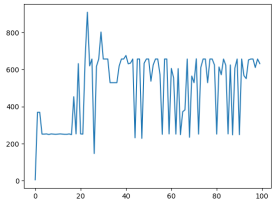


Fig. 8: Rewards gained by the agent during the training of the Dueling DQN