DeepRL Project: FlappyBird

Souhaiel BEN SALEM, Charbel ABI HANA, Adrian GARNIER, Israfel SALAZAR
{souhaiel.ben_salem, charbel.abi_hana, adrian.garnier_artinano,
israfel.salazar}@ens-paris-saclay.fr

1 Introduction

This assessment consist in implementing a tabular reinforcement learning algorithm to tackle the Flappy Bird game. We decided to use different deep reinforcement learning algorithms and modifications searching to achieve the maximum performance. We describe in this report the different approaches we took to solve the this task, making particular attention on the winning implementation but also mentioning why other implementations did not succeed.

This report will be organized in the following sections: In Section 2 we describe the game and its parameters. In Section 3 we quickly describe two attempts that were discarded because of the low performance, PPO and A2C. In Section 4 we describe our winning algorithm based on DQN. Here, we discuss the implementation and some variations of DQN, and the hyperparameter exploration to ensure the best performance. We finish in Section 5 presenting and discusing the results.

2 Description of the Game

Flappy Bird is a game published in 2014 for mobile devices. The objective of the game is to navigate a small bird through a series of pipes by tapping the screen to make the bird flap its wings and fly upwards, while gravity pulls it down. During the game, a point is given each time the bird passes over or under a pipe, and it looses a point if it collides with the pipe. The game theoretically has no end, but in this version, a message saying that the game was won is reached at around 20 pipes.

2.1 Environment Description

The environment is described by a tuple, the first value being the information about the bird, its x and y coordinate and its vertical velocity. The second element is a list with information about the pipes. Each pipe has 4 values, the x value for the left and right corners, the height of the pipe, and a Boolean indicating if it is coming form the top or bottom of the map where True indicates that the column is coming from the top.

2.2 Environment Transformation

In order to use DQN to estimate the reward, there are several changes we make to the data. The first change is adding the velocity in the x axis. This value is constant in this scenario, but it gives us the same shape of the pipes so it allows us to add it into the network in a single matrix. The second change is the pipe structure, we describe the pipes position in relation to the current position of the bird, so we define h and v to signal the horizontal and vertical distance from the bird to the pipes edges. Finally we place a 1 or 0 to indicate if its coming out of the top of the bottom, and an additional value to indicate if it is present or not. We had no need to include the width of the bar since in this scenario it has a constant width of 0.1.

In order to use DQN, we need to create a fixed size of input, so we create a variable called Field of View (FoV) that indicates how many pipes it can see. Then we take the pipes, we sort them in order of horizontal proximity, and then we remove all the pipes that have passed the Dropping point. We decided to make the dropping point the point in which there is no possibility of the bird colliding with the pipe. To generate the final description, we take the first N pipes, and if there are no more available, we fill up the rest with padding and we indicate that they are not in the screen.

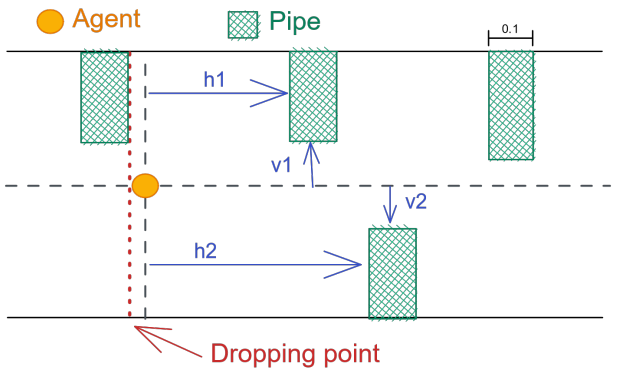

Figure 1: Illustration of the different variables of the environment.

3 Other Attempts

3.1 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) algorithm is a state-of-the-art reinforcement learning (RL) method that combines the benefits of trust region policy optimization (TRPO) with the simplicity of first-order optimization algorithms.
The key contribution of PPO lies in the optimization objective, which includes a clipped surrogate objective function. This clipping technique ensures that the policy updates remain within a trust region, preventing excessively large updates that could harm the learning process. By maintaining a balance between exploration and exploitation, PPO achieves stable learning while reducing the likelihood of policy degradation.

3.1.1 Intuition

We decided to try PPO on our Flappy Bird environment due to its several advantageous properties. Some of the reasons for selecting PPO :

- Robustness: PPO has demonstrated robust learning capabilities across a diverse range of tasks, including both continuous and discrete action spaces. Flappy Bird, with its discrete action space, could benefit from the algorithm's robustness.
- Stability: PPO's clipped surrogate objective function helps maintain stability during training by constraining policy updates within a trust region. This prevents overly large updates, which could destabilize learning in the Flappy Bird environment.
- Sample efficiency: PPO is an on-policy algorithm, which is generally more sample-efficient than off-policy methods we tried earlier. In our Flappy Bird environment, this characteristic can lead to faster learning and potentially better performance with fewer training iterations.

3.1.2 Challenges and Limitations of Applying PPO in our environment

Although we initially believed that PPO would be well-suited for the Flappy Bird environment, we encountered several issues that limited its effectiveness. First of all, our Flappy Bird environment has sparse rewards, which can make it challenging for PPO to learn an effective policy. PPO relies on gradient-based updates, which can be hampered when rewards are infrequent and hard to relate to actions. Additionally, PPO requires fresh data for each update whereas in the our environment, with its highly dynamic and potentially short episodes, the agent may not accumulate enough meaningful experience before needing to update its policy. This also makes finding the right hyperparameters challenging to which the performance of the PPO algorithm is sensitive.

3.1.3 Possible solutions

To make PPO work in our Flappy Bird environment, one can try several RL adaptation techniques such as reward shaping which consists of introducing an intermediate rewards to guide the agent's learning more effectively. For example, we can provide small rewards for maintaining a certain height or distance from the pipes, or for staying alive for a longer duration. This can help the agent receive more frequent feedback and potentially speed up the learning process. Another approach is frame stacking where we can combine several consecutive observations into a single input for the agent which can help our agent better understand the dynamics of the game and learn to anticipate the movement of the bird and pipes.

Another approach that PPO could benefit from is curriculum learning, that consists of training the agent with simpler versions of the environment and gradually increase the difficulty. For example, we can begin with larger gaps between pipes or slower pipe movement and progressively make the game more challenging. Howver this approach would require more training time and episodes which defies the constraints of the problem.

3.2 Advantage Actor Critic A2C.

We also tried implementing Advantage Actor Critic, A2C. This method searches to reduce the variance in the reinforcement learning response combining policy-based and value-based methods. We learn two approximation functions: a policy that controls the agent and a value function that indicates how good the action taken by the policy is. A2C is designed to optimize the performance of an agent in an environment by maximizing the expected cumulative reward. It achieves this by estimating the value function of the current state and using it to calculate the advantage of each action taken by the agent. This advantage is then used to update the policy and value functions, allowing the agent to learn the optimal policy for the task at hand.

A2C seemed to be a good option for the Flappy Bird environment because it can learn a policy directly from the agent's interactions with the environment. Since the game has sparse rewards, the agent receives a reward only when it passes through a pipe, value-based methods like A2C can be useful because they estimate the value of each state, which can help the agent to generalize and learn from sparse rewards more effectively. However, this implementation failed mainly because of the time needed to search for the optimal hyperparameters. The convergence of the two networks make the algorithm more sensible to the initialization and finding the optimal parameters to win the game was too complex.

4 Deep Q-Network

The Deep Q-Network (DQN) works by having a network that learn how to estimate the reward of a given state. There are 2 main components for this approach, a deep neural network and a replay buffer. First, the deep neural network has an input that represent the state of the environment, and an output of the expected Q value for each action. Additionally we have a replay buffer that keeps track of previous episodes played, and this allow us to sample actions from earlier with a random order. The replay buffer allows us to remove the correlation between the transition in the batch, allowing the network to learn properly.

In our case, we are using a Convolutional Neural Network as the network from the agent. This consist on two convolutional layers, and one fully connected linear layer of 64 neurons, and finally an output for

each move. Figure 2 shows a diagram of the architecture.

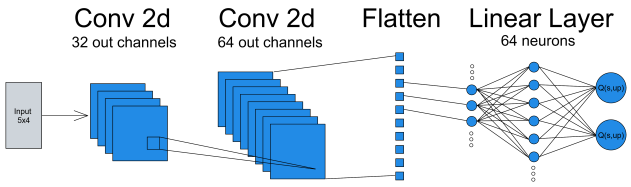

Figure 2: Diagram representation of the DQN Network used.

For the training we found out that reducing the exploration over episodes had a good impact on the learning capabilities of the network for this game. In order to do this we indicate which value epsilon should start in, which will be the final value epsilon can take, and in what episode it should get to that point.

Another change that was tested was changing the schedule over the network parameters. Three scenarios were tested, updating after 1, 10, and 100 episodes.

4.1 Dueling DQN

The Dueling DQN is proposed as an improvement over DQN. This proposes that the Q-value can be divided into $V(s)$ which is the value of a given state, and $A(s,a)$ which is the advantage of taking one action over another one. You can obtain the Q-value of an action by adding the advantage and the value of the state $Q(s,a) = A(s,a) + V(s)$ since A indicates the impact a certain action would have within the given state, being positive or negative. The benefit that this architecture provides is a faster converging time, this was shown on the Atari benchmarks [Wang et al., 2016].

In order to use Dueling DQN, we used the model shown in 3. This model has the same convolutional layers as the DQN model we previously showed, but instaed of just having one linear layer of 64 neurons, it has two. The lower part of the model creates an initial estimate of Up and Na, and the top part generates $V(s)$, Finally, to get the final output we do the following operation $Q(s,a) = V(s) + A(s,a) - \frac{1}{N}\sum_k A(s,k)$.

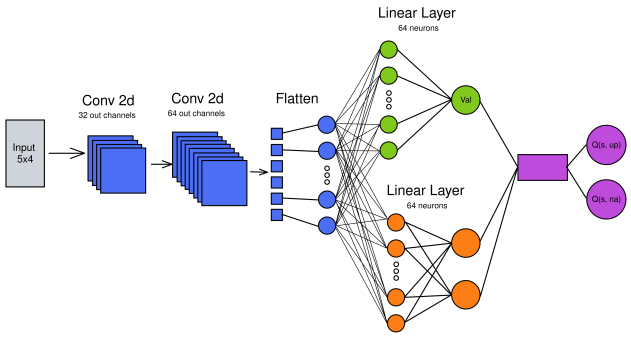

Figure 3: Diagram representation of the Dueling DQN Network used.

This final step is important to guarantee that the Dueling DQN learners the values as expected. In the paper, it says that in order for this approach to work, we have to force the network that gives the advantage to have an average of 0. One of the ways we can enforce this is by getting the final outputs, and subtracting the mean from the vector, this way the sum of the values will be equal to 0.

4.2 Double DQN

Another problem with the DQN is that it tends to overestimate the reward of a given state, action pair. [Van Hasselt et al., 2016] demonstrated that the base DQN tends to overestimate the values for Q which is harmful for the training and the overall performance sometimes resulting in a sub optimal policy. The cause of this tends to come from the max operation in the Bellman equation:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, a)$$

Where $Q'(s_{t+1}, a)$ were the Q values calculated by the target network. The authors of the paper proposed to extend DQN separating the action selection and action evaluation into two different networks that estimates the Q-values. One of the networks, the "target network", is used to estimate the Q-values, while the other network, "online network" is used to select the actions. This implies, choosing actions for the next state using the trained network but taking values of Q from the target network. Using this, the new expression for the target Q-values will be:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q'(s_{t+1}, arg\max_a Q(s_{t+1}, a))$$

4.3 Dueling Double DQN

This agent essentially combines both extensions; we use the dueling DQN network architectures and the optimizer updates the loss function developed from the double DQN agent. We use two separate networks for both Double and Dueling DQN with one network used for selecting actions and another for estimating the Q-values. We also split the Q-value function into state value and advantage for each action, allowing for efficient learning of valuable states and advantageous actions.

5 Results

The training procedure consisted of doing sweeps over the hyperparameters of the 4 DQN agents and we obtain the following combination: updating the target parameters at larger steps showed less stability during training, using the scheduler on the ϵ parameter showed more effective as the training was stabilized throughout the episodes. A batch size of 32, learning rate of 3×10^{-4} and a discount factor of 0.8 showed the best results. On these same hyperparameters, for maximum training episodes of 1000 and evaluating at each 10 epochs where each validation step was on 100 episodes, we show the results obtained in figure 4.

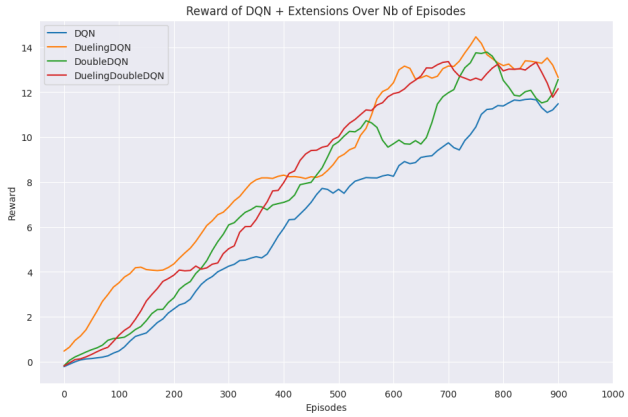

Figure 4: Rewards on DQN base and Extensions.

In the figure above we can see that all the DQN methods have a similar learning curve. However, we can clearly see that extensions of DQN not only achieved a higher performance on the game but also converge faster. This fast convergence is particularly marked for DuelingDQN. In genereal we observed that DuelingDQN and DuelingDoubleDQN obtained the highest scores.

We show that for 100 validation episodes, the dueling double DQN model obtained an average reward of 25.42 with a standard deviation of 22.4. That was the best model we trained and we show the comparison with the baseline model (Stable Agent) (figure 5) which had an average reward of 3.68 and standard deviation of 5.53 and the base DQN model with an average reward of 24.34 and a standard deviation of 24.2. From interacting with the environment, on runs where we obtained an average reward of more than 20 the environment showed a success message which meant that the problem was solved.

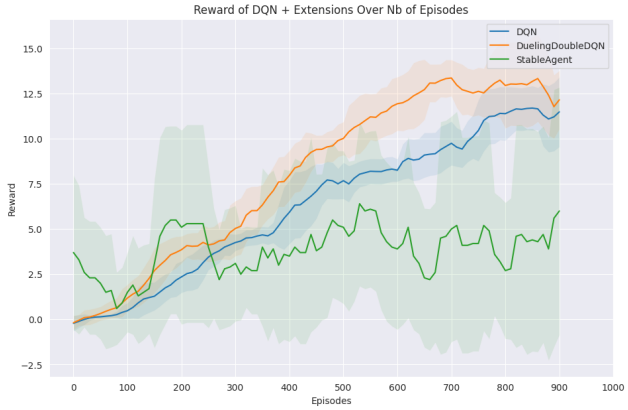

Figure 5: Rewards on DQN base and Extensions with Stable Agent.

6 Conclusion

In this project, we successfully solved the FlappyBird environment through the DQN method and further optimized it with hyperparameter sweeps and implementations of extensions of the DQN method namely Double, Dueling and Double Dueling DQN. We obtained the best results on the double dueling DQN model with the lowest variance in the reward and the highest mean reward. Training showed good reward convergence but we could further improve the stability of the network through bigger networks and longer training time (we had to adhere with a maximum of $2h$ of training). We could also explore more extensions of the DQN method such as a prioritized replay buffer developed in [Schaul et al., 2016] where sampling from the replay buffer isn't uniform but samples are assigned priorities according to the training loss. Noisy DQN in [Fortunato et al., 2019] also demonstrated a very simple idea for learning exploration characteristics during training instead of having a separate schedule related to exploration through adding noise to the weights of fully connected layers of the network and adjust the noise through backpropagation. Rainbow DQN in [Hessel et al., 2017] combined all of the above extensions and got the most improved DQN model which we can explore further in this project.

On the other hand, Proximal Policy Optimization (PPO) and Advantage Actor Critic (A2C) faced challenges in handling the sparse rewards and dynamic environment of Flappy Bird. Despite these difficulties, these algorithms have potential for future improvements, such as through reward shaping and curriculum learning, as discussed earlier in the report.

One interesting aspect to consider for future work is the generalization capabilities of the trained agents. In our experiments, we focused on training the agents in a specific Flappy Bird environment. However, it would be valuable to investigate how well these agents can adapt to new, unseen environments, such as modified Flappy Bird levels or even different games with similar dynamics.

References

[Fortunato et al., 2019] Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2019). Noisy networks for exploration.

[Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning.

[O'Shea and Nash, 2015] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.

[Schaul et al., 2016] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning

with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

[Wang et al., 2016] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.