RL Flappy Bird

Hugo de Rohan Willner

CentraleSupélec, Gif-sur-Yvette, France

1 Experimental Setup

1.1 Environment Description

The environment used is a simplified version of the Flappy Bird game. There are two versions available. The first version returns as an observation the complete screen render of the game, whereas the second returns the distance of the player from the center of the closest upcoming pipe gap along the two axes (x,y).

The main limitation of the complete-screen version is that it is the state is much more complex (of dimensions screen width x screen height), whereas the distance version space is much simpler (2 dimensional). Training of the complete-screen version may therefore take much longer than the feature-engineered distance counterpart. In this version, the input dimensions will also change if the screen dimensions change. The limitation of the distance version is that the features may not be expressive enough to train a high performing agent. Furthermore, an advantage of the complete-screen version is generalisability: once a training pipeline has been setup, it can be potentially be used to train an agent on other games (if the screen size and pixel values are the same).

Since the Orginal Flappy Bird gym environment also provides a distance version, a trained agent could potentially be re-used, although further training may need to be done due to small discrepancies such as the pipe width being more than 1 pixel. However, the Original full-screen version would not work since it is a 3 channel RGB version, where the pipe and bird and not explicitly given.

1.2 RL Algorithms used

Two RL algorithms are implemented: Monte-Carlo control (MC) and SARSA-Lambda (SL).

2 Results

2.1 Episode Capping

After getting some first results with uncapped episode lengths, we notice some unstable behaviours from the agent, as shown in the graph below.

In the uncapped version shown in green, the average reward increases during training, but the agent often gets episodes with very low rewards.

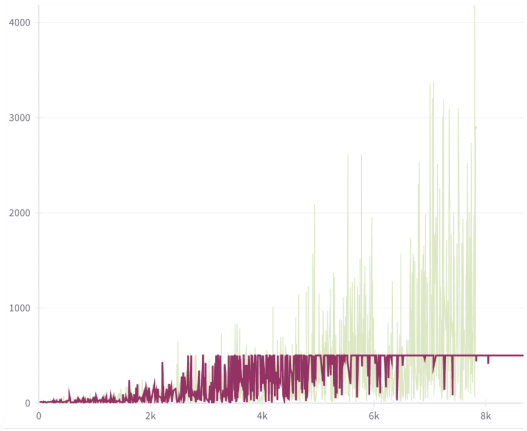

Fig. 1. Total episode reward vs episode - Comparison of capped and uncapped episodes

Outputting the policy at the wrong step might return a low performance agent. In the version shown in red, episode lengths are capped at 500. We observe that the agent learns to consistently get the maximum available reward at each episode at the end of training. Due to the repetitive nature of this game, we can expect an agent that performs well on 500 steps to also perform well on any amount of steps. This cap is therefore applied to the rest of the results.

2.2 Hyper Parameter Tuning

Hyper parameter tuning was undertaken using a Bayesian search. Since the final reward value is capped at 500, I chose to optimise the average of the total episode reward over the training run. This will allow us to find the agent which learns the fastest. The results of this seach for both algorithms is shown below:

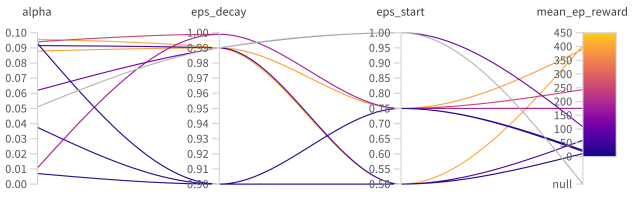

Fig. 2. Monte-Carlo: Param Values vs Average Reward

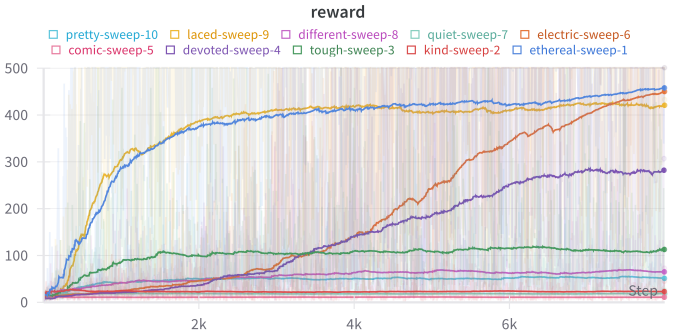

Fig. 3. Monte-Carlo: Average Reward during training

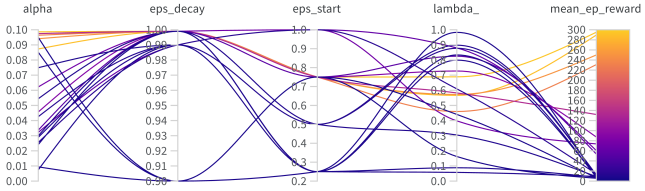

Fig. 4. SARSA-Lambda: Param Values vs Average Reward

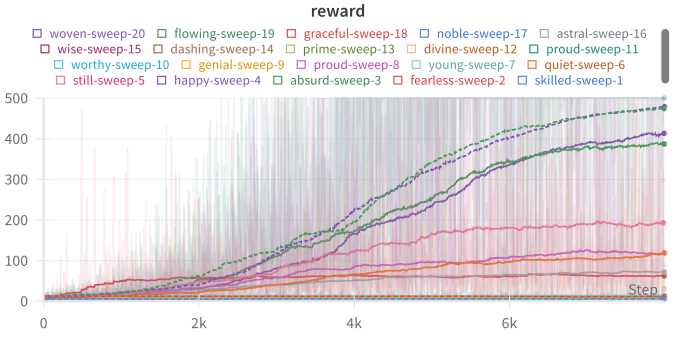

Fig. 5. SARSA-Lambda: Average Reward during training

2.3 State Value Function Plot

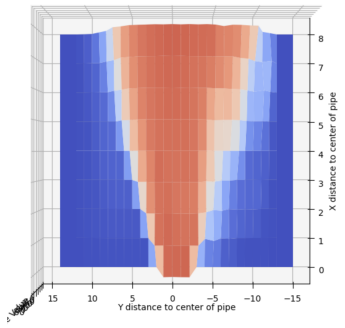

Fig. 6. State Value Plot after training

The figure above shows that as the bird gets closer to the pipes (x distance), it becomes more important to be at the level of the gap (y distance).

2.4 Discussion

During the parameter search, we notice that SARSA-Lambda is more sensitive to the hyper-parameters used. This can be explained by the fact that MC uses the explicit return for Q, whereas SL bootstraps. The bootstrapping has to be done correctly for the method to work, hence it is sensitive to the parameters controlling this.

2.5 Notebook Link

`https://github.com/hugodrw/RL-FlappyBird`