Flappy Birds

Tobias Glaninger, Ahana Deb

The code used in this work can be found here- /https://github.com/ahanadeb/Flappy_birds

1 Game Environment

Here we explore the game Flappy Birds, and compare the performance of different reinforcement learning approaches on this game. The objective of the game is to make the bird fly from left to right without colliding with the pipes equally distanced horizontally, and the vertical gaps are generated randomly. OpenAI gym environment consists of

- observed state space
- action space which allows interaction with the environment and returns a new state from the environment,
- reward associated with the state-action pair.

A standard screencap from the game environment is given in 1

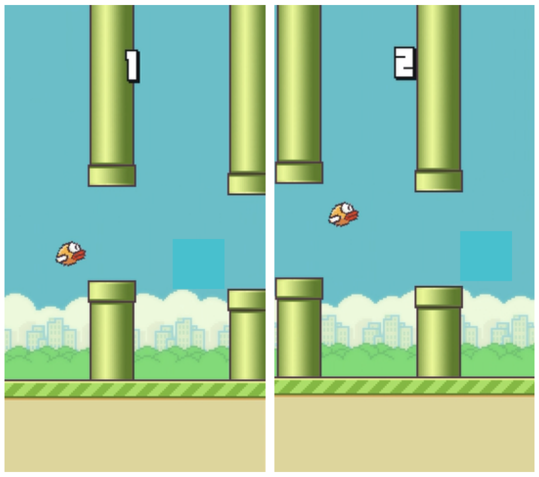

Figure 1: Flappy-bird OpenAI gym environment gameplay screencaps.

1.1 Action space

In this game the action space is uncomplicated, with only two discrete actions at each time-step, choosing to fly upwards, or not doing anything at all, allowing the bird to fall downwards. These two actions are denoted by 1 and 0 respectively.

1.2 State space

Flappy Birds has been explored as learnable RL environments quite extensively, this repository (Lin) by Yen-Chen Lin here uses DeepQ learning on this game to get impressive results. Their approach includes first removing the background of the environment, to obtain images as shown in 2, and use consecutive convolutional and maxpooling layers and a penultimate layer of fully connected ReLU nodes to obtain a final output layer having dimensions equal to the number of actions in the action space, in this case 2.

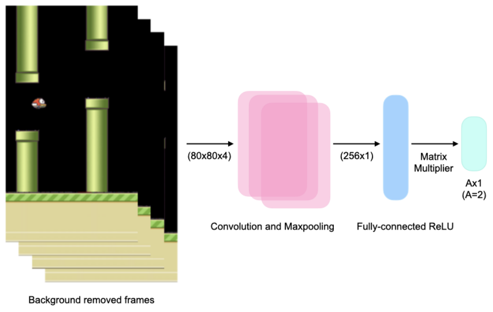

Figure 2: DeepQ approach to Flappy bird (Lin).

(Chuchro) also approaches the game in a similar way, by first reducing the background of the environment but additionally preprocesses the game frame to obtain a further simplified representation as shown in 3.

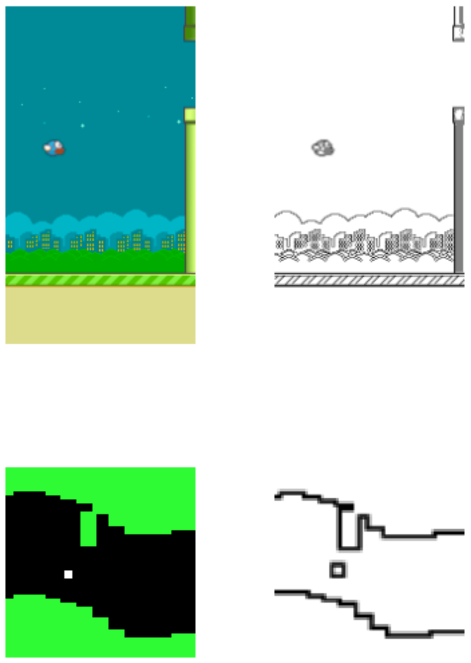

Figure 3: Simplified approach to Flappy bird (Chuchro).

1.2.1 Our approach

In our approach, we try both the previous approaches of treating the entire matrix of RGB pixel values as our state space, and also, owing to the simplicity of the game, the state space can be condensed to only two variables, the vertical distance from the upcoming gap between the two pipes and the horizontal distance from the pipes. These two variables, as we'll see in the results are enough to encode the state of the game, and select the right action. The state space in this case is the range of pixels on the screen, and the cardinality is therefore very high, so we discard tabular approaches for this case.

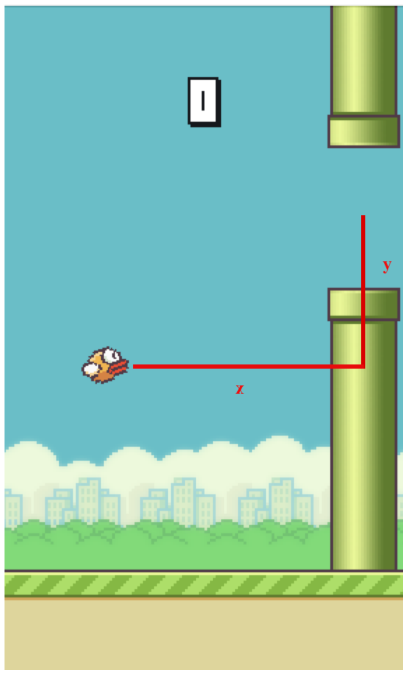

Figure 4: The horizontal distance from the pipes, and the vertical distance from the gap are taken as the two state features. (Chuchro).

1.3 Reward

The game originally offered a reward +1 for crossing a pipe gap successfully, but we modified the reward function to

$$reward = 0.01 \cdot t + 10 \cdot game["score"] - state[1] \qquad (1)$$

where t is the current time of continuous flight, $game["score"]$ is the number of pipes successfully crossed, and $state[1]$ is the distance along the $y - axis$ from the bird's current position and the position of the gap. We also penalise crashing on the ground with a very high negative reward of -100. Removing the dependency on the time of continuous flight, during training, causes the bird to crash often on the ground, even before it discovers the high reward awaiting the crossing of the first pipe. The particular weights associated with each reward are what we found work empirically.

2 Results

Both (Chuchro) and (Lin) have experimented on this environment using DeepQ learning, with impressive outcomes, so we wanted to explore other algorithms would perform in this scenario. Our initial approach included TD-learning and Sarsa (shown in 7), and despite training for 50,000 episodes, the first pipe obstacle couldn't be reliably cleared. This is also owing to the original design of the game environment; when the bird clears the right side boundary of the pipe, the score updates to a high value without waiting for the bird to cross the left side boundary of the gap. So the algorithm does discover a reward for "clearing" the first pipe, and sticks to it.

We also tried Actor-Critic (2) to solve this which gave much better results as seen in 7 and it learn to reliably cross the first pipe within the first 1000 episodes of training.

The pseudo-code for the algorithm we used is given as

Algorithm 1 Actor Critic pseudo code

```
1:  Initialisation of parameters $\gamma$, learning_rate, random_seed, NN for prediction of value function and policy
2:  for i_episode ... MAX_EPISODE do
3:      for $t = 1 \dots T$ do
4:          Sample action from policy using the current state $\pi(s)$
5:          Sample state and if the bird crashed from environment
6:          Calculate reward according to chapter 1.3
7:          Accumulate reward of step to reward of episode
8:          Break if bird crashed and game is over
9:      end for
10:     Back-propagate loss through NN for prediction of policy
11:     End training if three consecutive runs had a higher reward than a threshold
12: end for
```

The average reward per length of the episode is shown in 7. As observed, in the initial episodes, for both sarsa and actor-critic, the agent struggles to cross the first pipe, and around 1000 episodes, when it first discovers the reward associated with scoring a point in the game, the average reward obtained by the actor-critic increases.

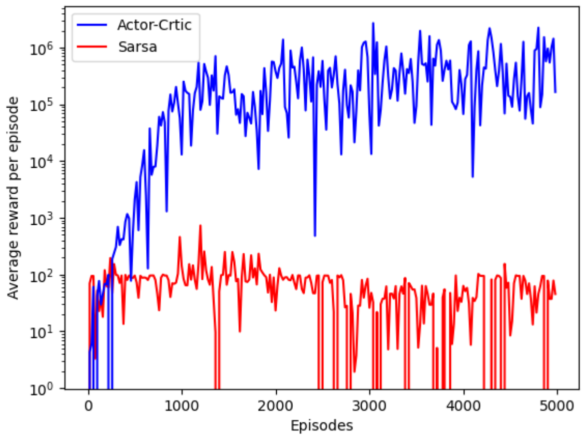

Figure 5: Average rewards received per episode for offline Actor-Critic (blue) and SARSA (red).

6 shows the episode lengths as a function of the number of episodes trained on. While Actor-Critic reaches the highest limit of timesteps we allowed, Sarsa could only reach 10,000 timesteps before colliding with the generated pipes.

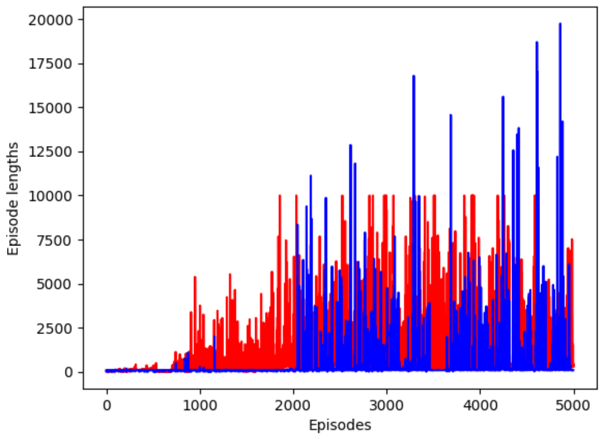

Figure 6: Episode lengths before termination with number of episodes run, for offline Actor-Critic (blue) and SARSA (red).

Fig [] shows the comparison between the rewards accumulated per episode for different sizes of hidden layer used in the Actor-Critic network. Comparing intermediate layer sizes of 2,10 and 64 for 5000 episodes of training, we can see that the network of 2 size learns practically nothing, whereas of size 10 takes much higher number of episodes (compared to our original 64 size layer) to discover the first rewards beyond the pipe.

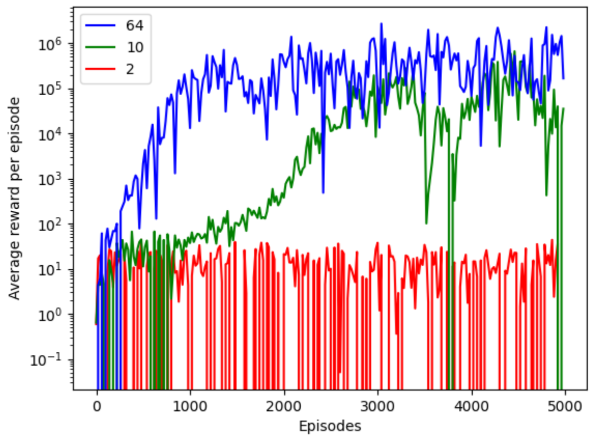

Figure 7: Average rewards received per episode for different sizes on Actor-Critic network

3 Limitations

We wanted to attempt the hardest version of the game (fastest velocity of the bird and smallest gaps between the pipes), and the main limitation we faced was working with the complete RGB environment of Flappy Birds as our entire state space. Even after training for 10^7 episodes, our agent couldn't figure out getting past the first pipe. We also tried a background-removed version of Flappy bird with similar results (except for DeepQn, although in this case it did get past the first few pipes. Working with an easier version of the game (increasing the gap between the generated pipes, keeping the velocity of the bird constant) led us to much better results for both actor-critic and sarsa.

Further improvement to our current approach can be made by adding the distances in x and y direction of the second pipe on the right side, as the main reason for crashing in our solution is situations in which the pipes have a big difference in their y coordinates. These circumstances can only be solved by flying through the first pipe in a certain angle.

References

[Chuchro] Chuchro, R. Game Playing with Deep Q-Learning using OpenAI Gym. http://cs231n.stanford.edu/reports/2017/pdfs/616.pdf. [Online; accessed 20-April-2023].

[2] Konda, V. and Gao, V. (2000). Actor-critic algorithms.

[Lin] Lin, Y.-C. DeepLearningFlappyBird. https://github.com/yenchenlin/DeepLearningFlappyBird. [Online; accessed 20-April-2023].