Reinforcement learning - Individual assignment

Antoine Poupon

CentraleSupélec, Gif-sur-Yvette, France

1 Experimental setup

I chose to train the 2 agents on the environment that returns the distance of the player from the center of the closest upcoming pipe gap, as it seemed more advantageous, as we will see in the next part.

To control exploration, we use an epsilon-greedy exploration strategy, with a decaying epsilon to ensure the experiences get more and more relevant along the time. The epsilon decay drastically impacts the learning curve of the agent because if we take a coefficient very close to 0, it will exploit the policy it learnt late in the experiment which is not something we want as we would like the convergence time to be small. We can observe this phenomenon with both agents.

In general, the convergence time is a lot smaller when training the Monte Carlo agent. It is due to the fact that we do many computations at each timestep when training the Sarsa(λ) agent whereas we only do some of these computations (such as Q-value update) when training a Monte Carlo agent.

Due to limited budget, I didn't run the training of the agents for more than 6000 episodes but it could be interesting to check when the optimal Q values are reached using both algorithms.

2 Environments and their limitations

To facilitate further discussion, we call *environment 1* the environment that returns the complete screen render of the game and *environment 2* the environment that returns the distance of the player from the center of the closest upcoming pipe gap. In *environment 1*, the agent possesses complete knowledge from the conventional game screen, whereas in *environment 2*, it focuses solely on a surrogate objective rather than considering all visual elements of the game. The surrogate objective is defined in a way that makes it sufficient for the agent to accomplish the task at hand.

The main limitation of utilizing *environment 1* is that if the configuration of the TFB environment were to change, the agent's policy might struggle to generalize. This is because the states it was trained on may no longer resemble the current environment state. It's different for *environment 2* because the agent's policy, once trained, doesn't directly depend on the configuration of the environment.

Another limitation of the *environment 1* is for policy visualization. The fact that the state space of the *environment 1* is in two dimensions facilitates the visualisation of the policy, whereas it's more complicated with the *environment 1*.

3 Adaptation of our agent to original flappy bird environment

An agent trained on the *environment 1* cannot be used in the original flappy bird environment (in particular the version "FlappyBird-rgb-v0") because its policy has been trained on game screen encoded as integers which is very different from images. To get a policy that can be efficient on image states we would need to use a Convolution Neural Network policy because a simple dictionary Q cannot store all images the agent has seen with low computational cost.

An agent trained on the *environment 2* could be used in the original flappy bird environment (using the version"FlappyBird-v0") as well, because even if the state space of the 2 environments are similar, the one we used in this project is discrete whereas the one used in the original environment seem to be continuous.

In conclusion, we would need to use function approximation to get a working policy on the original flappy bird environment, which we could get using algorithms such as Deep Q-Network or Policy Gradient algorithm.

4 Change of configuration for TFB environment

By changing a bit the configuration, we can see that the agent still makes rational choice when the pipe gap is not too far from it but is random when it's too far. The more we change the configuration, the more random the agent will be.

5 Results and performance of the different agent

5.1 Monte Carlo Agent

Hyperparameter optimization First we do hyperparameter tuning using Bayesian search to trained our agent properly. We use the Weight and Biases API and runs our experiments with 10 differents sets of hyperparameters.

Note that we do hyperparameters tuning using the average reward since the beginning of the experiment as optimization objective. I made this choice because I want to optimize the final policy as well as the convergence time but it could be a good idea to train an agent and only use the reward obtained by exploiting greedily the final policy, as it is was really characterize a good or bad agent once trained.

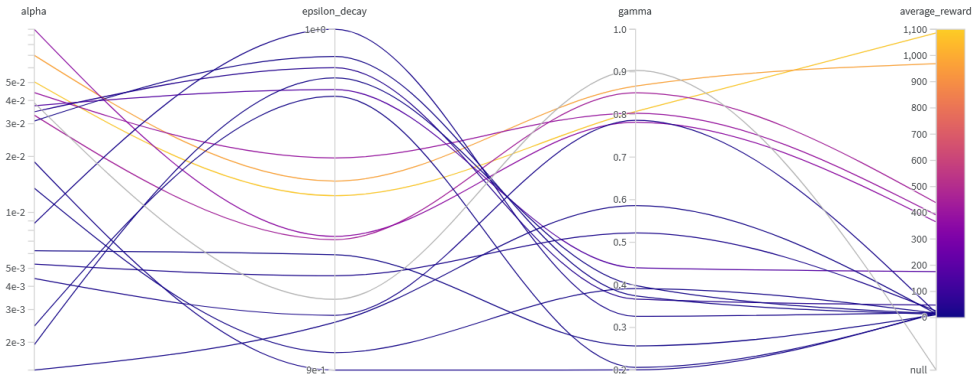

Fig. 1. 18 sweeps of Bayesian Search

We find that the best set of hyperparameters is:

- Step size α : 0.05
- Discount factor γ: 0.95
- Epsilon decay : 0.8

State value function Here is a plot of state value function of the Monte Carlo agent. We can see that the more the bird is aligned with the center of the pipe gap, the higher the state value is. If we runned many more episodes, the red color should be homogeneous, and the reddest state should be the one at y = 0 from the center of the pipe gap.

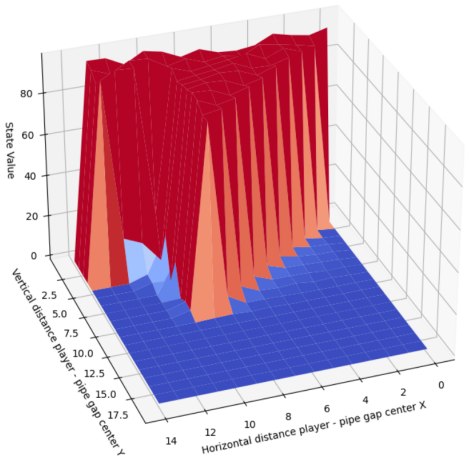

Fig. 2. State value function V(s)

5.2 Sarsa(λ) Agent

Hyperparameter optimization Using the same methodology as for the MC agent, I find another set of hyperparameters that don't perform very well. By tuning the hyperparameters by hand, I get the following hyperparameters that gives satisfactory results:

- Step size α : 0.03
- Discount factor γ: 0.95
- Epsilon decay : 0.999
- Trace decay : 0.98

State value function We get the following state value plot:

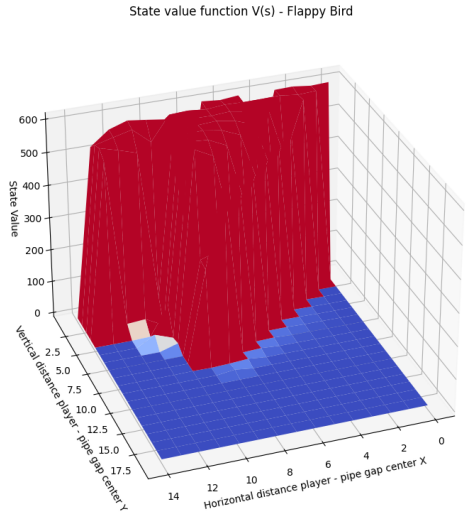

Fig. 3. State value function V(s)

We notice that the state highlighted in clear red in Figure 2 is depicted in blue in Figure 3. This indicates that the SARSA(λ) agent exhibits a more conservative approach, assigning lower state values to states that may not appear optimal.