CentraleSupélec

RL Agents for Text Flappy Bird

Quentin Macé

April 12, 2024

Contents

1 Experimental Setup

The code used to make this report is available at : **This repo**

1.1 Environnement setting

For all the experiments below, unless stated otherwise, we will be working with two environnements :

- "base" environnement : state is $(distance_x, distance_y)$, height=15, width=20, pipe gap=4
- "screen" environnement : state is a "screenshot" (a 15*20 pixel boxes where values can be either 0, 1, 2). height=15, width=20, pipe gap=4

1.2 Repository structure

The bulk of the agents are coded inside the agents.py file (to avoid having a really big notebook)
To avoid redundancy, some base classes were also defined in the same file. In it there are 3 main abstract classes, 1 MC control agent class and 1 Sarsa lambda (+ QLearningAgent + ExpectedSarsaAgent + DeepQLearningAgent, but i will not dive into those in the report) :

- Agent class : base class for all agent, initialises the paramters that are in common for all the agents and some basic functions (argmax, getPolicy...)
- OnlineAgent class : class for Temporal differences based agents, defines a train function which is in common for all the agents of this type (I also tested QLearning agents and ExpectedSarsa agents although I will not talk about it in the report)
- OfflineAgent class : class for Monte Carlo based agents, implements the generateEpisodeFromQ method and the train method, which is in common for all agents of this type.

There is also a utils.py in which i defined plotting functions and the grid search function used to find the best hyperparameters.
Appart from that all the experiments were made in the experiments.ipynb file.

2 Results on text environnements

Firstly, because of computational limitation, I could not run on very large grid search on all parameters. In the hand, i ended up testing 30 40 combinations for each grid search I did while arbitrarily fixed others. More details will be given in each subsection.

2.1 Monte Carlo control

In MonteCarlo control we update the Q value at each episode by first generating an episode until the end, keeping record of the different rewards and states. The update rule is:

$$Q_{t+1}[state_i, action_i] = Q_t[state_i, action_i] + \alpha(G_t - Q_t[state_i, action_i])$$

Here is the search space explored by the grid searches:

- α (step size for Q value updates) $\in \{0.01, 0.1, 0.5\}$
- ϵ (probability of random action at the start of training) $\in \{0.1, 0.5, 0.9\}$
- ϵ_{decay} (value by which is multiplied ϵ at the start of each episode) $\in \{0.9999, 0.999, 0.99\}$
- ϵ_{min} (minimum value for epsilon during training) : fixed to 0.01
- γ (discount value) : fixed to 0.9

2.1.1 Base Environnement

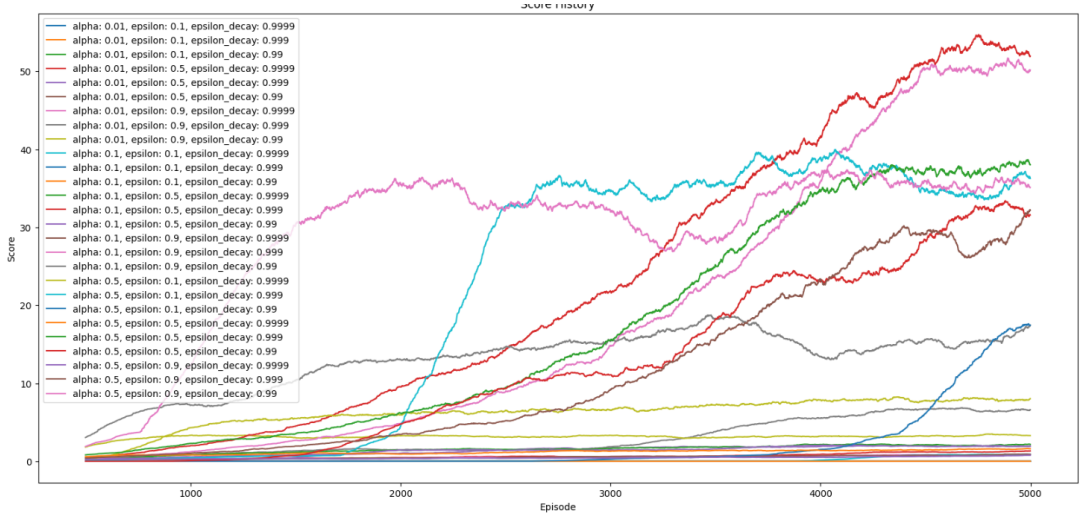

Figure 1: Scores history for Monte Carlo control agent with different hyperparameters in base environnement

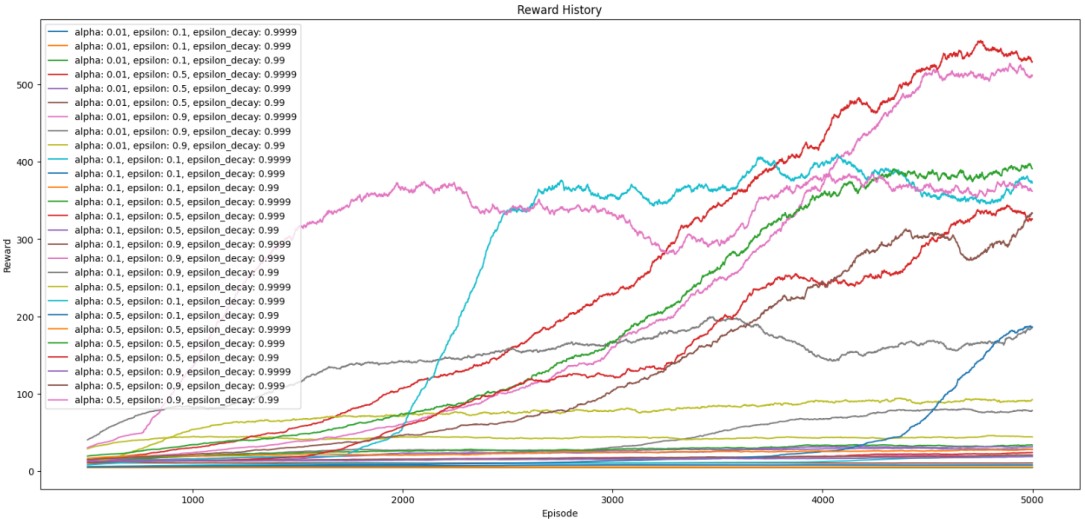

Figure 2: Reward history for Monte Carlo control agent with different hyperparameters in base environnement

We can see that performances can vary quite a lot depending on the set of parameters used. Experimentaly, the runs were particularly sensitive to both *alpha* and $epsilon_{decay}$. This can be easily interpreted as α is very much related to the "learning speed" of the agent, thus too high values of alpha will lead to poor convergence properties, while low values will leed to slow convergence.
In the case of the sensitivity to ϵ_{decay}, it shows how important the tradeoff between exploration and exploitation is important in reinforcement learning.
The best run here was obtained with the set $\{\alpha : 0.5, \epsilon : 0.5, \epsilon_{decay} : 0.99\}$ (red curve at the top from 1 and 3) both in terms of final mean score value and in terms of progression rate throughout the training.

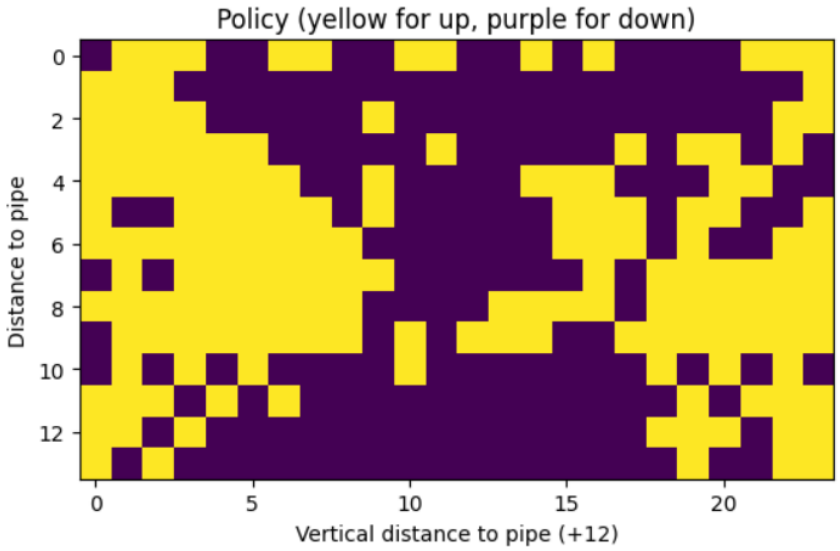

Figure 3: Policy of the best MC control agent

We can see that the agent has learned (while not perfectly) that it is better to flap when the bird is below the pipe and not too close to it. (when it is really close to it, the policy is not important since the agent while undoubtedly fail). The value function plots are also available in the notebook.

2.1.2 Screen Environnement

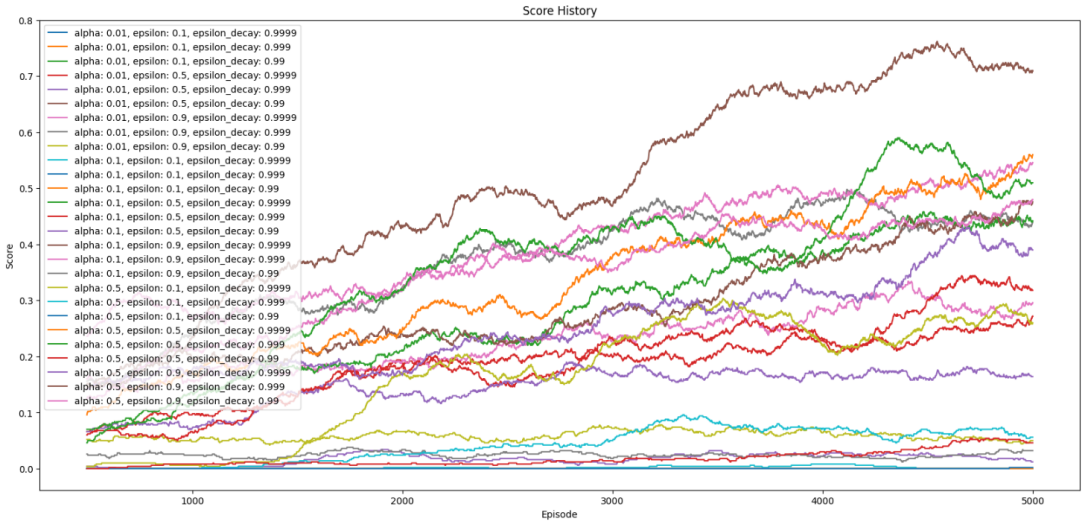

Figure 4: Scores history for Monte Carlo control agent with different hyperparameters in screen environnement

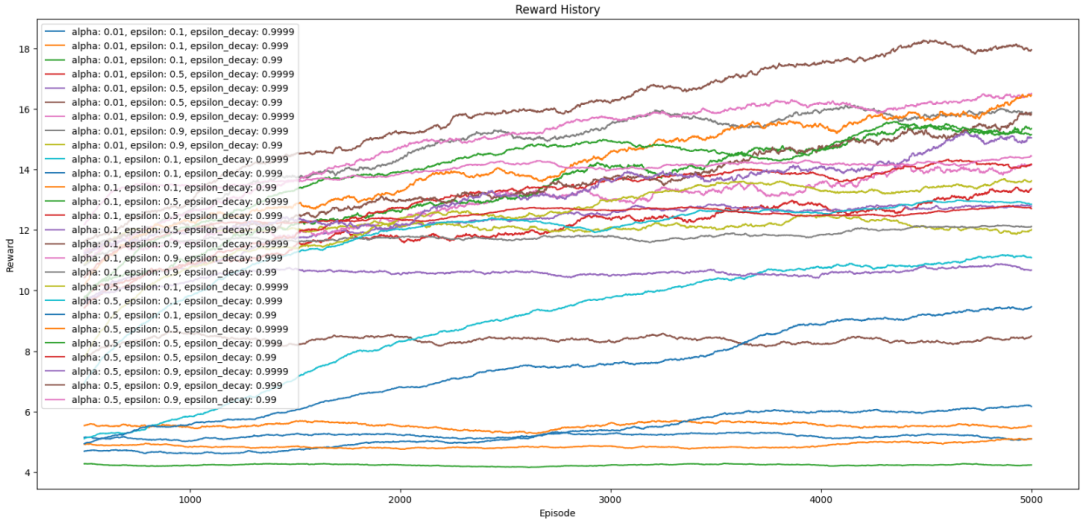

Figure 5: Reward history for Monte Carlo control agent with different hyperparameters in screen environnement

As could be expected, with the same number of episodes, the scores and rewards are much lower when it comes to the screen environnement.
Indeed, the information about the state is much more complex (not just 2 integers, the number of possible states goes from 200 to 3^{300}), so the task is much harder to learn.
The best set of parameters found during grid search is $\{\alpha : 0.5, \epsilon : 0.9, \epsilon_{decay} : 0.999\}$ (brown curve at the top from 4 and 5) I also ran an experiment where I kept the obtained best parameters and let the training run for a much longer time (on 50000 episodes). It is still not anyway near the scores obtained with the base environnements. The state space is too large for any tabular approaches as we are doing here.

2.2 Sarsa(λ)

The Sarsa lambda agent code is described in the section 12.7 of the reinforcement learning book. However, i implemented a slightly different version explained at this link

2.2.1 Base Environnement

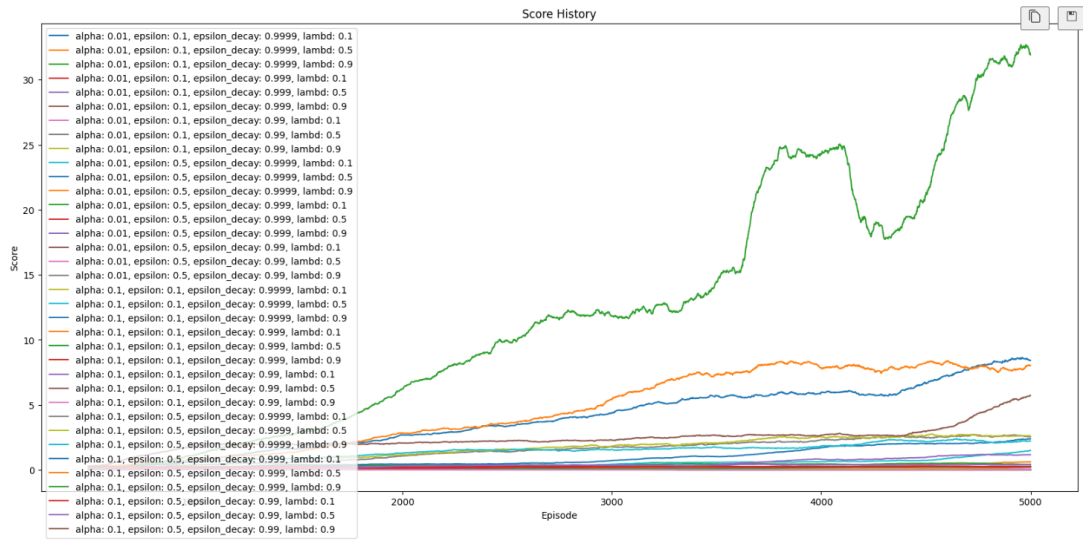

Figure 6: Scores history for Sarsa(λ) agent with different hyperparameters in base environnement

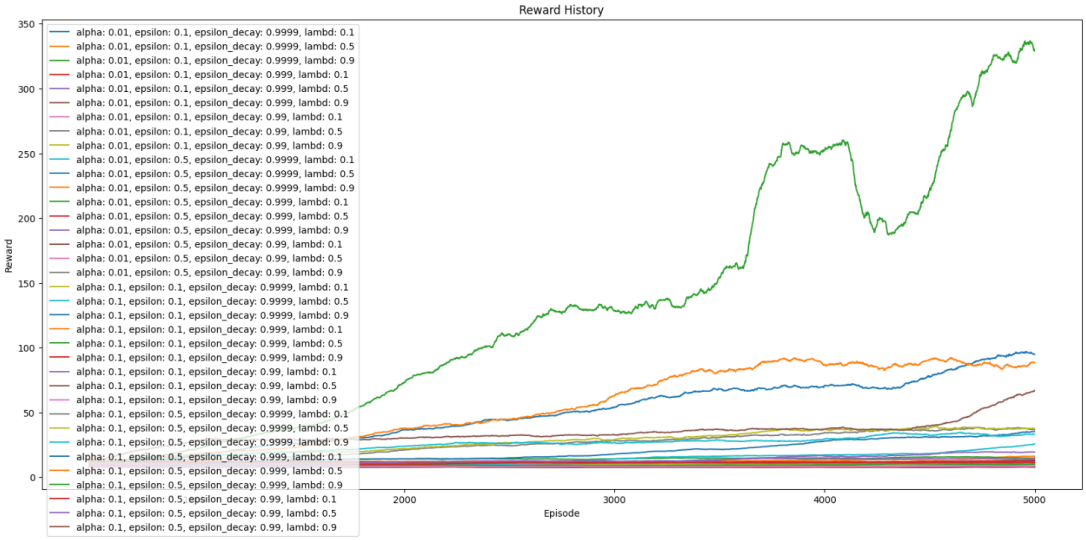

Figure 7: Reward history for Sarsa(λ) agent with different hyperparameters in base environnement

We can see that the best agent was trained with the parameters : $\{\alpha : 0.1, \epsilon : 0.5, \epsilon_{decay} : 0.999, \lambda : 0.9\}$, and it is the best from quite far.
This underlines that Sarsa(λ) appears to be quite more sensitive to parameter changes than Monte Carlo Control. Since the best set of parameters beats the other by quite a huge margin.
Also, we can see that our best agent didn't quite converge yet, the convergence time for Sarsa(λ) seems quite higher than the one for MC control.

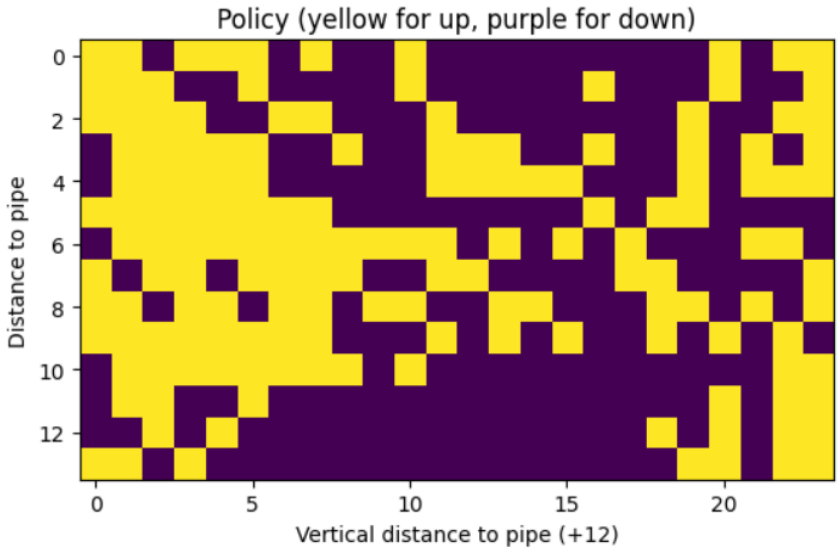

Figure 8: Policy for the best Sarsa(λ)

The Policy for the agent is quite similar to the one obtained with MC control agent (and that is reassuring since they both learnt the same game !). It is better to jump on the left side of the graph (when we are below the hole).

2.2.2 Screen Environnement

I also (like for MC control) tried training agents on the screen environnement, however the results are the same as for MC control, the state space is just to huge for the agent to learn anything. (the graph is available in the notebook).

2.3 Possibility to generalise to the original flappy bird environnement

It should not be possible at all for those agent to generalize to the original flappy bird environnement. Indeed, for the text environnements, the state space is discrete, thus a tabular approach can work. But in the case of the original environnement, the state space is continuous (or at least enormously large). Thus there is no hope for a tabular approach to work in this setting.
I implemented a DeepQLearning agent in order to solve this problem but had no time testing it, still, it is in the agents.py file (commented out).

2.4 Generalisation to other settings

In the last section of the notebook, I experiment with generalisation to other settings. For that i chose two additional environnement (each time the "base" environnement, given the scores it would not have done anything with the "screen" one). The experiments were lead with the best MC control agent.

- **easy** setting : {height = 17, width = 23, pipe-gap = 5}
- **difficult** setting : {height = 13, width = 17, pipe-gap = 3}

Since the state spaces are different for the two new environnments, and our agent use a tabular approach, we can't simply used the policy given by our agent on the original envrionnement.
My approach is : if the state does not appear in the original policy, choose a random action.
Given this setting, here are the results for the agents generalisation :

Environnement	Average Score	Average Reward
Original environnement	31.639	329.108
Easy environnement	13.628	174.621
Difficult environnement	1.336	23.024

Table 1: Average scores and rewards for our best agent on the original environnement

As we can see, as soon as we change a bit the settings, the performances decrease quite a lot in any case. However, the agent manages to generalise quite well to the easy setting compared to the hard one. This can be easily understood since all the actions that were good for the original environnement should be good for the easy one. The performance degradation should only come from random actions on unseen states.
This is not true anymore for the difficult setting since the pip gape decreased. Thus the performance is really bad if the agent does not learn and adapt its policy.