Reinforcement Learning Flappy Bird Assignment

Thomas Cosyn

Abstract. Training simple Reinforcement Learning agents in text-flappy-bird gym environment.

1 Agents

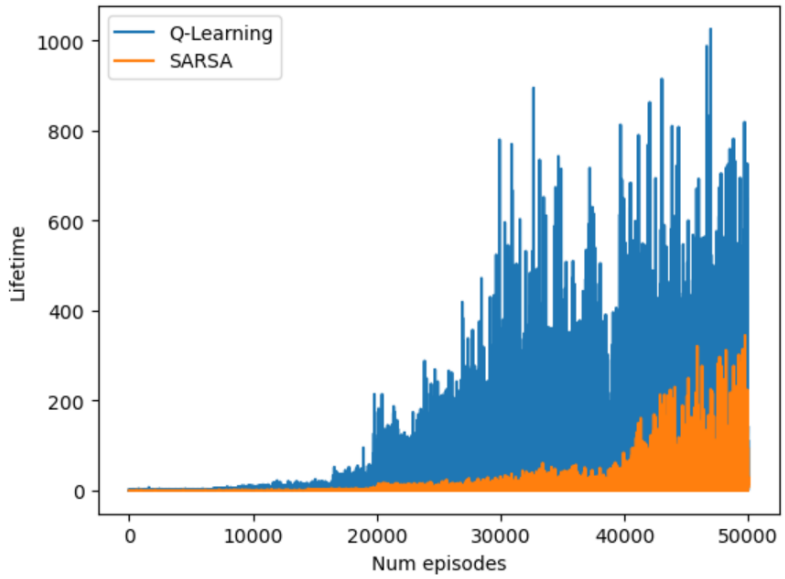
We are asked to train an agent to hit the highest score in the TextFlappyBord-v0 gym environment. We can formulate this as a Reinforcement Learning problem:

- The observation of the agent is the couple of coordinates (dx, dy) describing the agent's position. dx is an integer that ranges from 0 to 11, and dy is also a signed integer ranging from -15 to 15.
- The state as implemented in the environment is the tuple (observation, info) with info being a list containing the score and distance to next pipe.
- The action space is simple, it is a binary value, being 0 for idle and 1 for flap.

Since our state-action space is quite small and discrete, the simple algorithms seen in class will be enough, thus we choose to train :

- A SARSA agent
- A Q-Learning agent

To compare the two agents, we set their parameters to the same values ie $\epsilon = 0.01$, $\alpha = 0.1$, $\gamma = 0.95$ with no epsilon decay, and we look at the evolution of the score when training :



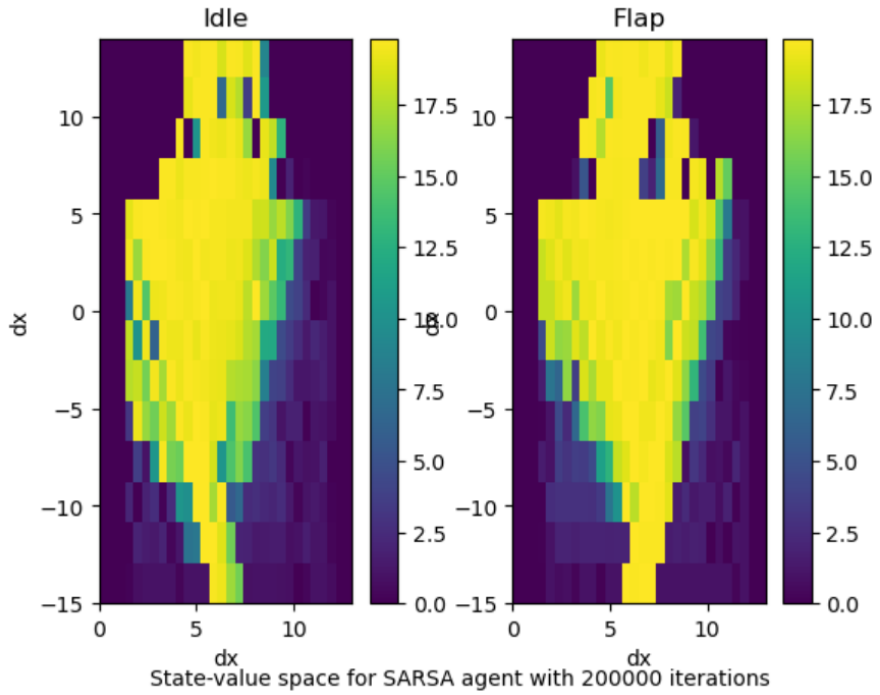
We clearly see a faster convergence for Q-Learning. We also notice that the performance of both of my algorithm have a high variance, since they have low scores even after a high number of episodes. This is probably due to the partial randomness of the epsilon greedy policy, we will then try to add an epsilon decay in a later part, to correct this.

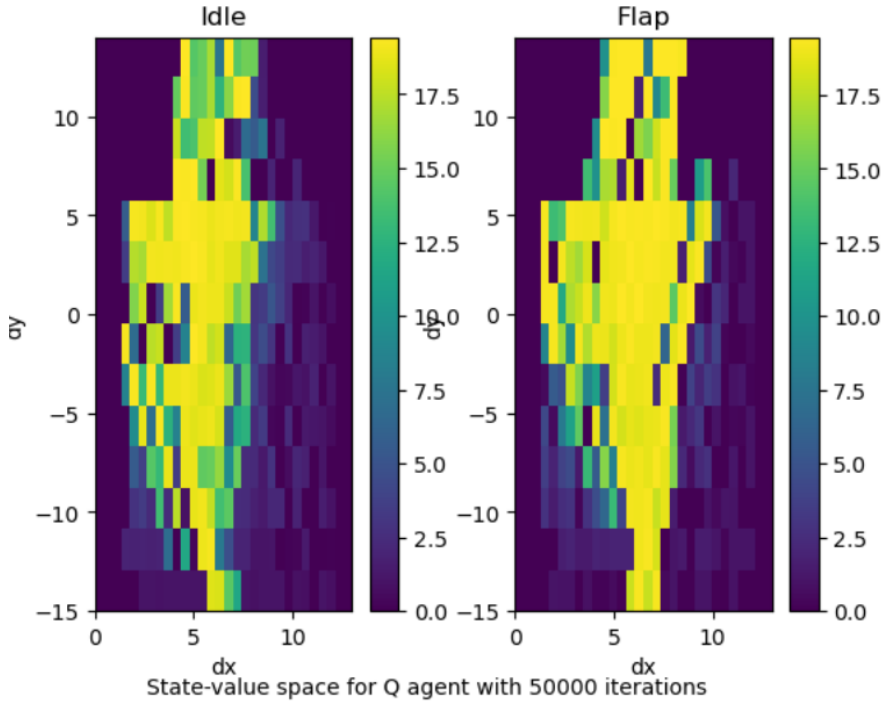
2 State-value plot

Once our agent has learnt, we can visualize the state-value plot. Here the state is a couple of coordinates, so we can easily visualize such a plot using a heatmap.

We can see on the figure below that the best state for our agent is to be at the top of the pipe gap, to let himself fall into the gap. We also see a certain symmetry when comparing the state-value plot for actions either “idle” or “flap”.

This result is the same for both Q-Learning agent and SARSA agent. We can then conclude that both converges toward the same state-value space, but as we saw in part 1, Q-Learning converges way faster.


Idle
Flap
dx
dy
dx
dx
State-value space for SARSA agent with 200000 iterations


Idle
Flap
dy
dx
dx
State-value space for Q agent with 50000 iterations

3 Parameter sweep

To study how the agents behave when sweeping parameters (ϵ, γ, α), we need to define a relevant metric.

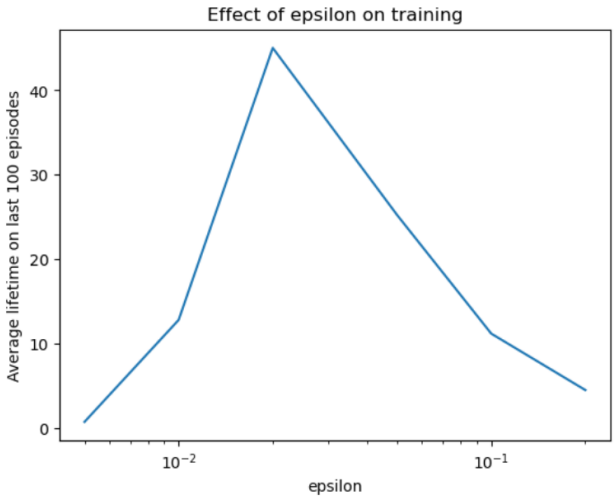
Since we can easily train a model that will never lose, (and that I want don't want my experimentation to be computationally expensive), the interesting aspect is more about the convergence speed.

Thus, we will compare what is the average score on the last 100 games of the agent during is training, when training `50000` episodes.

Thus the method will be to sweep parameters on a logarithmic scale and to see the evolution of the average score obtained on the last 100 episodes of the training. I now that to be more precise, I should make several experimentation for each parameter value and average (that is what is done in research paper) but I really don't have the resources and time to do that.

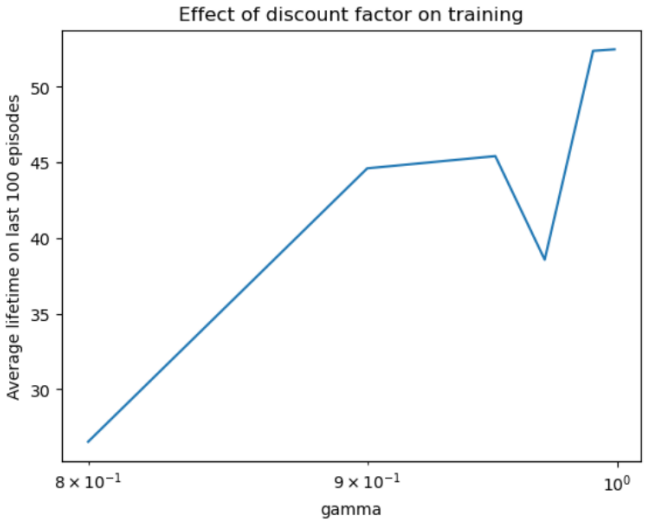
We do this study by fixing all the parameters and sweeping one to see the influence of this one, but in an optimization context or a Kaggle competition, we could have also grid searched the parameters.

3.1 Epsilon



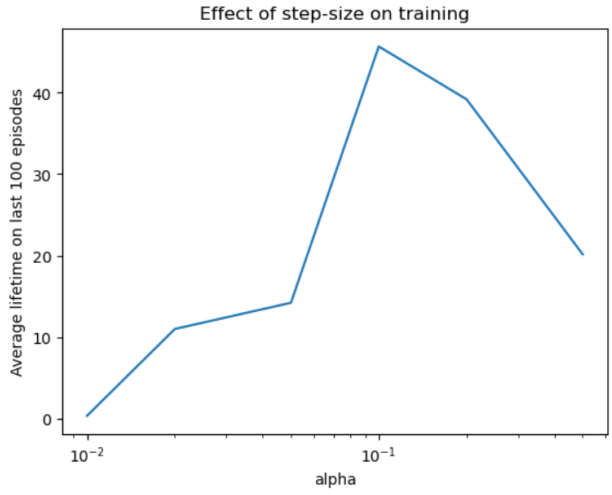
The convergence speed of the agent is very sensible to the epsilon value, as a too low value leads to a weak exploration of the environment and high values prevent the agent to exploit the policy he has learnt, by taking stupid random actions.

3.2 Discount factor



Sweeping the discount factor clearly shows that the agent better be far-sighted, ie with a discount factor close to one. It means that a decision at one step has an important impact on the reward obtained at the next steps. It is quite logical, getting away from the pipe gap even many steps before reaching it increases the probability of the death of the agent.

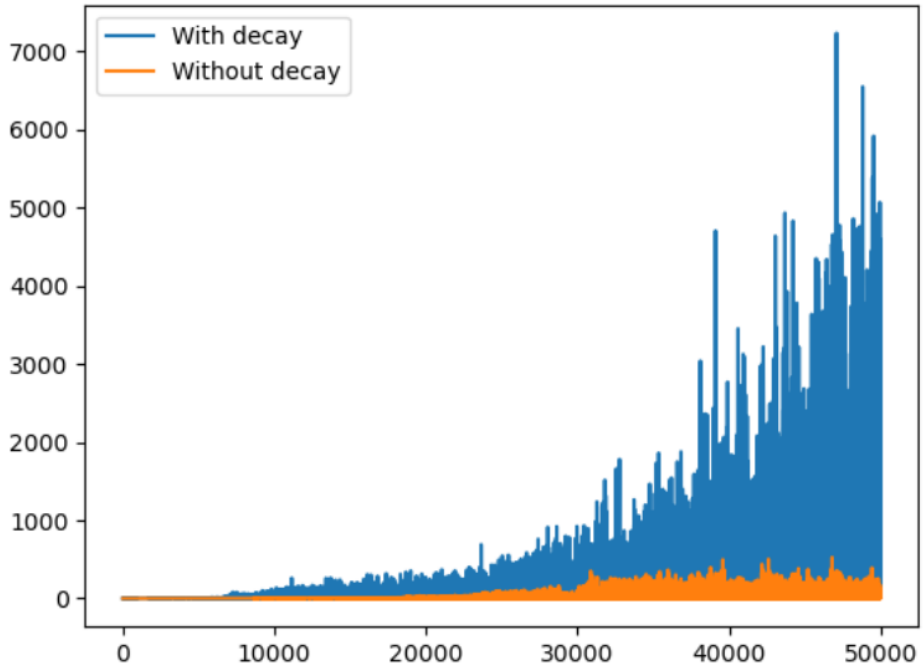
3.3 Step-size update



It appears that the step-size also has a strong influence on the convergence speed.

3.4 Epsilon decay

As seen before, at some point, the estimation of the state value is good enough and taking random action becomes pointless. To cope with that, we can implement an epsilon decay, it means, updating geometrical epsilon to decrease it towards zero as our agent learns the environment, here is the evolution of the scores during training, with and without epsilon decay for the same set of parameters.



Indeed, using the decay trick clearly improves the convergence speed.

4 Transposing the agent to others environment

It is interesting to see if we can use our agent trained in the simplistic text-flappy-bird environment to others, more sophisticated environments.

4.1 TextFlappyBird-screen-v0

The main difference between TextFlappyBird and TextFlappyBird-screen is the observation space. The latest allows the agent to be in a full information environment, ie instead of observing only its own position, the agent visualizes the whole environment.

It is then required to train another agent considering this specificity. Having more information, this agent would then be better since it would be able to anticipate the next pipes.

4.2 Original game

In the same way as with TextFlappyBird-Screen-v0, the observation space is not the same. In the environment the observed position is composed of two float coordinates instead of integers. Thus to use our agent in this environment, we would need to implement a transformation that takes the float coordinates of the game environment and gives the equivalent as a couple of integer coordinates. This method amounts to defining a grid on the environment as seen in class.

To get an even better agent, we could then implement grid overlapping, or retrain an agent on the continuous action space using Neural Network Value Approximation, or on the shelf algorithm such as Proximal Policy Optimization from the stable-baseline library.

References

1.