UNIVERSITÀ DEGLI STUDI DI PADOVA

Homework #3 Deep Reinforcement Learning

Monaco Saverio - 2012264

Neural Networks and Deep Learning - Professor: A. Testolin

***Abstract*—The final homework concerns Deep-Q Learning, an algorithm of Deep Reinforcement Learning based on *Bellman equation* for decision processes.**
Two environments of OpenAI Gym are tested with the Deep-Q Learning algorithm: "CartPole-v1" where a 2-dimensional platform must balance straight a pole, and a custom environment of a popular mobile game "FlappyBird".
For the latter, two different state-space encodings are probed, a smaller state that employ 2 distances variable, and a bigger one that represent the game visual output.

DEEP-Q LEARNING

Deep-Q Learning algorithm belongs to the class of Deep Learning problems of Reinforcement Learning. In this branch of Deep Learning, an agent can interact with the surrounding environment (simulated or real). By the resulting environment it is possible to give the agent a reward, namely a parameter that it tries to maximize.
The goal of this algorithm is to find an optimal *policy*, namely a rule that tells what action to make, given the state of the system. The way to determine the *policy function* is not unique, Deep-Q Learning relies on *Bellman equation* with the hypothesis that every instance is a Markov Process.

For both environments, two optimizations are implemented to the vanilla algorithm:

- *Experience Replay:* The agent does not learn while it acts on the environment. It follows the last policy for an arbitrary number of episodes, then it updates its policy sampling from its past [state, action] pairs. This helps the convergence of the algorithm by having a more i.i.d like data samples.
- *Target Network:* Two similar networks are implemented for a more stable update rule

I. CARTPOLE-V1

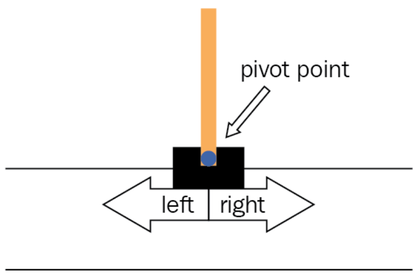

Fig. 1: CartPole-v1 graphical representation

A. *Methods*

B. *Results*

II. FLAPPYBIRD

A. *FlappyBird-v0*

*1) **Methods**:* The state space are the two parameters of this environment, namely:

- horizontal distance to the next pipe: range = [-∞,+∞]
- difference between the player's y position and the next hole's y position: range = [-∞,+∞]

While the possible actions are just two:

- Fly upwards (originally the tap of the screen): `action value = 1`
- Do nothing `action value = 0`

Regarding the reward and punishes, the agent takes a point for every frame it is alive, and it gets a punishment proportionally to the vertical distance of the next hole.

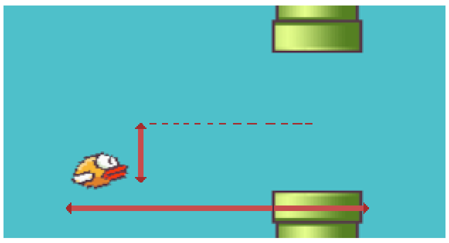
Fig. 2: State space of FlappyBird-v0 environment

a

Being highly similar to the last environment, it was built another network class called just `Flappy` with only some minor tweaks to the class already built. Likewise the last task, the hyperparameters were tuned using Optuna:

Parameters for Optuna Optimization:

Trials:	
Epochs:	
Lr:	
Optimizer:	
Gamma:	
Initial T:	

During the parameter optimizations, the policies of the model were tested every 10 episodes (at T=0) to check if it already learned how to properly play. The criterion for a policy to be satisfying is to get a mean score greater than ??? over 10 games.

*2) **Results**:*

Best parameters:

Learned in:	??? epochs
Lr:	
Optimizer:	
Gamma:	
Initial T:	

*B. **FlappyBird-rgb-v0***

*1) **Methods**:*

*2) **Results**:*

III. Conclusions

IV. Appendix