Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Deep Q-Learning: two case studies

Authors:
Mattia Fratello - 785142 - m.fratello@campus.unimib.it
Davide Meloni - 834333 - d.meloni5@campus.unimib.it
Alberto Raimondi - 771357 - a.raimondi21@campus.unimib.it

June 17, 2018

Abstract

This work aims to explore the recent field of Deep Reinforcement Learning and highlight some of its limitations and complexities. Starting from two environments available on the OpenAI Gym framework, two similar DQL models, although with some differences, are developed: both agents do not know at the start the space in which they are immersed, therefore the learning process is preceded by a phase of exploration of the possible states. It emerged that a simple neural network with few layers is able to create a sufficiently detailed policy so that the agent can understand how to maximize the reward, however, given the simplicity of the model used for not so easy tasks it would be necessary to implement a more precise phase of preprocessing and a more articulated architecture in order to obtain an optimal policy.

1 Introduction

Almost every machine learning progress you hear about (and most of what's currently called "artificial intelligence") regard supervised learning; in which the algorithm used is trained on a curate dataset that provides examples on which to learn. But in the last years another technique, reinforcement learning, is attracting the interest of many. Like so much of AI, reinforcement learning isn't new; the first textbook covering it dates to 1998 [1]. Reinforcement learning differs from supervised one cause here an agent learns by interacting with its environment. It isn't told to it by none what to do and it learns which actions to take to get the highest reward in a situation of trial and error, even when the reward isn't obvious and immediate. It learns how to solve problems rather than being taught what solutions look like. In an increasingly globalized world people, even if they live in the same country or if they share the same culture or social class, strongly differ among themselves for knowledge, abilities or desires. In a situation like this it's important, in the long term, to understand how an AI agent could be able to learn about these goals and about the peculiarities and abilities of the person it's working with and be able to personalize its assistance and actions to help that particular person achieve their goals.
The purpose of this project is to develop autonomous agents that, through reinforcement learning, are able to successfully deal with environments unknown to them at the start.

2 Environments

For this project two environments have been selected, offered by the OpenGym framework available in open source in python code [2]. OpenGym is an interface that provides various environments in which an agent can interact with the purpose of solving a problem through learning by doing. In particular, the two selected environments are the well-known mobile game Flappy Bird and an autonomous driving simulator called Car Racing and developed specifically to train reinforced learning algorithms. These two environments were used because they provide the image directly and not specific information on the state and this leads to use a more generalized algorithm that can independently select the necessary information.

2.1 Flappy Bird

The game consists in driving a little bird that moves continuously to the right, between a series of pipes. If the player touches the pipes or the edges of the image (top or bottom), he loses. The little bird moves upward each time the player touches the screen; if the screen is not touched, it falls due to gravity.

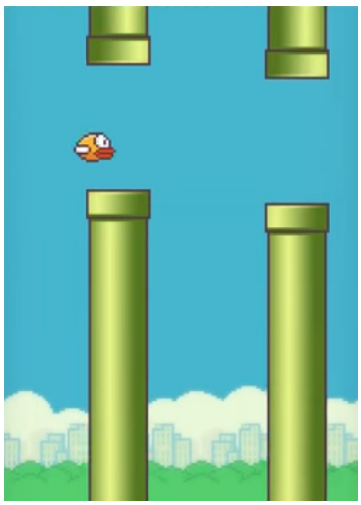

Figure 1: The image shows a typical state of the observations of the Flappy Bird game.

The observation made available by the environment is an RGB image in 288x512 pixel format (Fig. 1), while the action space consists of the simple choice at each step whether to move up or do nothing and then fall due to gravity. The reward is equal to +1 to each obstacle exceeded, -5 to each game over and nothing for each other step. It is therefore a discrete area of action, consisting of only two choices, and an extremely wide but not continuous observation space. Here the goal is to create a policy that allows the agent to decide the correct action to be performed according to the observation in order to overcome the largest number of pipes.

2.2 Car Racing

The environment of the Car Racing game consists of a car that must cover a track, drawn randomly in each episode. The track is inserted in a larger playing area and if the player exits from this area the game ends and the score obtained will be equal to -100.

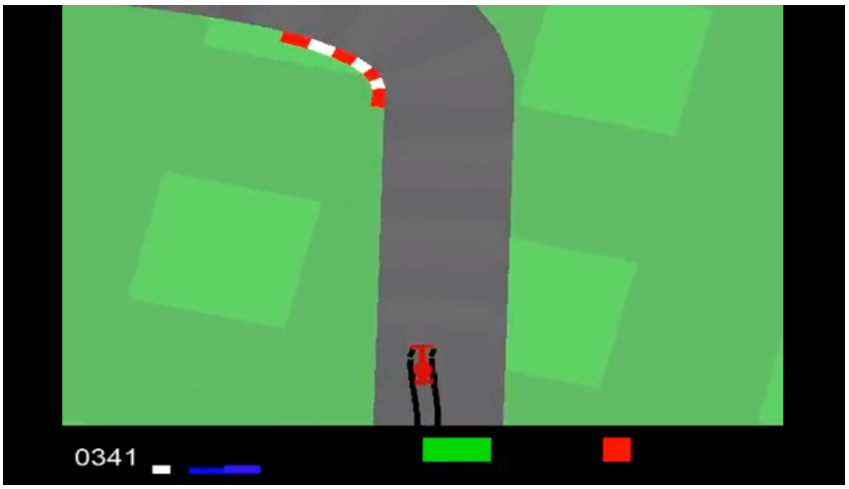

Figure 2: The image shows a typical observation of the Car Racing environment, the white bar refers to the vehicle speed, the blue one to the abs, the green one to the steering wheel while the red one to the gyroscope.

The observation made available by the environment is an RGB image in 96x96 pixels (Fig. 2), which in addition to showing the car on the track, displays some technical indicators (at the bottom of the image) related to

the vehicle such as: speed, four abs sensors, steering wheel and gyroscope. The action space consists of a larger number of choices than the previous case: the player can accelerate and/or brake by any value between 0 and +1, he can also steer right or left with values ranging from -1 to +1. The reward is equal to -0.1 for each frame and +1000/N for each section of the circuit visited, where N is the number of total sections on the track. Unlike the previous game, this is a continuous space of action, since each action consists of the combination of several choices to which continuous values can be attributed; the observation space remains extremely wide although not continuous. The objective is equal to the previous one, however in this case the correct action to be taken will be the combination of several choices e.g. accelerate of a value n + brake of a value n + turn right/left of a value n.

3 The Methodological Approach

3.1 Preprocessing

Flappy Bird Once the image is received from the Gym environment some preprocessing actions are necessary in order to present a more appropriate input to be used for the model. First, using the OpenCV2 library for python, the image is converted to black and white format and then a binary threshold function is applied that leaves only the contours of the relevant elements (bird and pipes, Fig. 3).

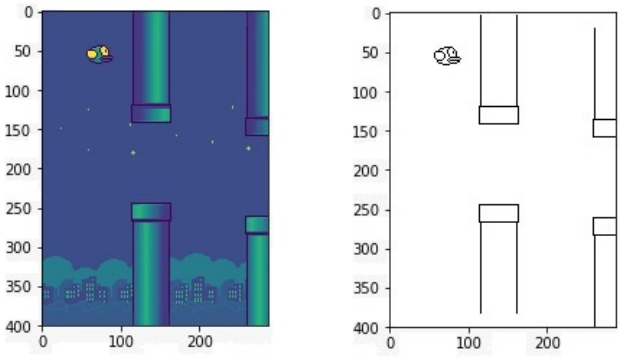

Figure 3: The two images show a typical observation made available by the Flappy Bird game environment before and after the preprocessing phase.

The image thus obtained is then analyzed in its various components:

1. the height of the bird is extracted from the pixel band relative to its vertical movements;
2. information concerning the height of the pipe is extracted from the extreme right band of the frame;
3. the proximity of the bird to the pipe is quantified with an iterative process ;
4. the vertical speed of the bird is obtained by comparing the current share of the bird with that of the previous frame;

Due to the imbalance between the effect of the two actions, caused by the fact that the jump action leads to a change in share more sudden than the null action, it was decided to give the possibility of choosing the action only every 5 frame; the intermediate actions are therefore all zero.

Car Racing Also for the Car Racing game environment some preprocessing actions are necessary. After transforming the image from colored to black and white (Fig. 4), it is divided into 3 distinct parts: the lower part, containing the indicators of the state of the car, is processed to extract numerical values from the dimensions of the bars; the upper part is resized in a 20x20 pixel matrix in order to obtain a more stylized image of the path section that the machine will have to do;

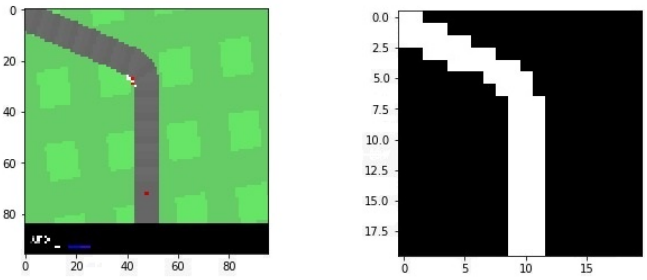

Figure 4: The two images show a typical observation made available by the Car Racing game environment before and after the preprocessing phase.

Finally, the image section relating to the vehicle (Fig. 5) is analyzed in such a way as to take into account its position and direction on the track.

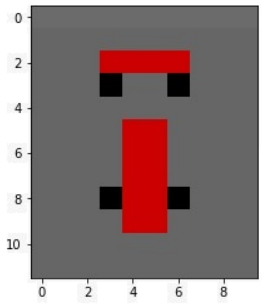

Figure 5: The image section relating to the vehicle after the preprocessing phase.

3.2 Deep Q-Learning Algorithm

The algorithm to delineate the policy begins with a phase of mere observation in which the episodes are performed randomly. For each epoch a tuple such as is registered:

$$Epochs[i] =< s, a, r, s' >, \tag{1}$$

where s is the initial state, a is the action performed, r is the reward obtained and s' is the state of arrival.

In the Flappy Bird game, after about 5,000 observation periods, the learning algorithm goes into action and at the end of each episode 32 epochs are drawn randomly from memory. For each initial and arrival state the Q-values relating to the two actions are predicted, then the Q-value of the starting state is corrected using the formula:

$$Q[s, a] = r + \gamma \, max(Q[s', :]) \tag{2}$$

The algorithm used for modeling the Q function is a neural network, a simple feed-forward model with 3 dense layers of gradually reduced dimensionality and an intermediate activation function called rectified linear unit (RELU) and with a linear activation for the output layer to obtain a continuous output (Fig. 6). The back-propagation process leads the Q-values to converge,

suggesting more and more accurately to the agent which action to perform. A ϵ parameter adjusts the randomness of choice of the action; as mentioned, for the first 5,000 epochs the development is totally random, so ϵ has been placed equal to 1, then the ϵ has gradually decreased until the action is completely discerned from the algorithm.

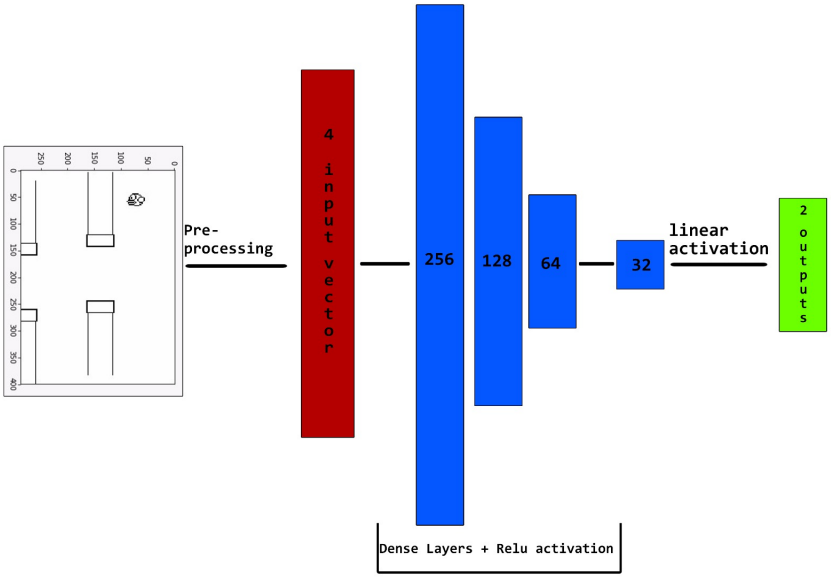

Figure 6: The image shows the Neural Network used for modeling the Flappy Bird's Q function.

Regarding the Q-Learning in the videogame Car Racing the algorithm is the same (Fig. 7) but the training process is done at the end of each episode and not taking advantage of a replay memory as in the case of Flappy Bird. A significant difference between the two approaches used is that the continuous actions space for the car racing environment was manually discretized to facilitate the backpropagation of the network, this means that the chosen actions are rarely normal and the car cannot obtain max score due to not being able to reach specific actions combinations.

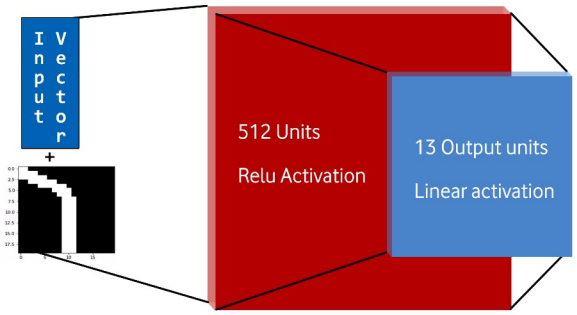

Figure 7: The image shows the Neural Network used for modeling the Car Racing's Q function.

4 Results and Evaluation

The plot in Fig. 8 shows the learning curve of the DQL algorithm for the Flappy Bird game. Furthermore, the table in Fig. 1 shows some interesting statistics of the training process and some results achieved by the agent. Equally the plot in Fig. 9 shows the learning curve of the DQL algorithm for the Car Race game. Furthermore, the table in Fig. 2 shows some interesting statistics of the training process and some results achieved by the agent.

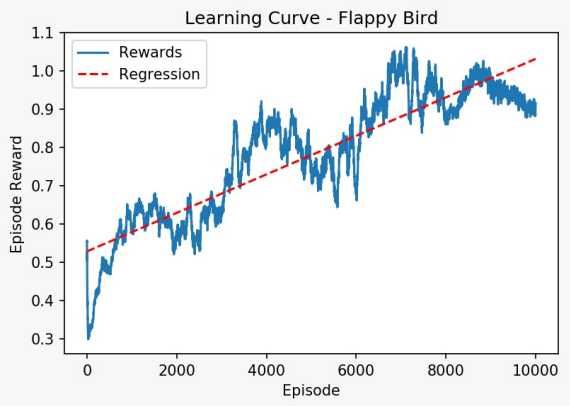

Figure 8: The image shows the learning curve obtained for the DQL agorithm used in the Flappy Bird environment. The red dash line is the linear regression of the data.

Table 1: Flappy Bird results

Number of episodes	10000
Average reward	0.78
Maximum reward	1.06
Regression line slope	0.00005
Execution time	5 hours

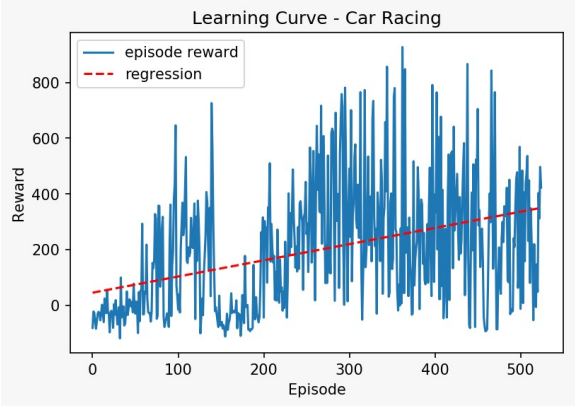

Figure 9: The image shows the learning curve obtained for the DQL agorithm used in the Car Race environment. The red dash line is the linear regression of the data.

Table 2: Car Racing results

Number of episodes	525
Average reward	197.5
Maximum reward	927.1
Regression line slope	0.582
Execution time	9 hours

5 Discussion

Flappy Bird The profile of the learning curve (Fig. 8) for the Flappy Bird algorithm shows a fluctuating trend, episodes with high results alternate with periods in which the agent can not overcome obstacles. This could be caused from several factors: it is supposed that the bird is not able to distinguish immediately when it is inside a pipe or not, indeed in the received observation it detects only the pipe closest to its right without taking into account what is above and below; this leads to a positive trend in overcoming the first obstacle but since the task is different for the next pipes the algorithm therefore fails to generalize the few examples introduced. Furthermore, it has been noted that the value of the randomness parameter ϵ strongly influences the training process of the algorithm. It is believed that the inability of the state to encode information about previous states is highly detrimental to the performance of the agent due to the fact that the previous choice of the agent are not known to him; this means that the used version of Flappy bird has not the Markov property and is difficult to solve with a simple dense neural network. The training examples fed to the network are unbalanced as they are mostly taken form the start until the first tube, due to this fact the algorithm is prone to initially overfit the weights to pass the first pipe but once the ability to pass it is achieved the weights start being fitted for passing the second pipe, a task that is different form the previous one but that the agent cannot recognize due to his limited viewing field.

Car Racing The results of the Car Racing game (Fig. 9) show a gradual but constant improvement of the policy until the average reward (for 100 episodes) of 400,then it starts plateauing; the average score obtained is particularly high considering that the task was pretty complex. The algorithm seems to recognize the track well and it is also often able to retake control of the vehicle after it has gone astray. It should be noted, however, that sometimes the car takes the route in the opposite direction, this confuses the algorithm because even if the agent is following the path no reward is given. The absence of the Markov property found in Flappy bird is partially canceled here by the fact that the underlying black bar gives us a lot of indirect informations about the previous actions taken by the agent.

6 Conclusions

The results obtained with the use of Deep Q-Learning algorithms have been positive even if not optimal. Both agents have effectively explored the environment identifying a good strategy for overcoming obstacles. However, there is ample room for improvement: for example,in the car racing environment a more sophisticated neural network could be implemented that, using an autoencoder to reduce the dimensionality of the image without losing the necessary information; LSTM (Long-Short Term Memory) cells could be used, recurrent neural networks that, given the current state and the action to be taken, are able to predict the next state. This type of architecture, proposed by [3], allows to infer the resulting states of all the possible actions for then choosing with greater accuracy through another network the best action to be performed. Regarding Flappy bird it is believed that a future approach using a convolutional neural network would greatly improve the results, as i would be able to autonomously analyze the image without preprocessing to find the position of all the tubes and the bird. This approach could be joined with a recurrent neural network to signal the agent the previous positions and actions of the bird making it more effective.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning - an introduction*, ser. Adaptive computation and machine learning. MIT Press, 1998. [Online]. Available: http://www.worldcat.org/oclc/37293240

[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[3] D. Ha and J. Schmidhuber, "World models," *CoRR*, vol. abs/1803.10122, 2018. [Online]. Available: http://arxiv.org/abs/1803.10122