Grado en ciencia e ingeniería de datos

Machine Learning I - Academic year: 2021/22

Reinforcement Learning

LUCÍA CORDERO SÁNCHEZ

ALONSO MADROÑAL DE MESA

NIAs:

100451778 100454449

Date: 13th May, 2022

1 getState function

To define the states we have kept in mind that simplicity was key. Having a small set of attributes for each state could make the difference between an enormous Q-table and a more discrete and manageable one. We have finally decided to create a new variable that would take advantage of the pre-existent ones and that would convey similar information using less code and creating less rows for our Q-table.

First we compute the distance between Pac-Man and all the ghosts, this just means selecting the closer one. The closest ghost will be our target, and when it is eventually eaten the program will recompute the distance and find the new closer ghost. After we have chosen the objective, we will focus on the relative position between it and Pac-Man, this means we will calculate in which direction we should move to find it, whether it is North, South, West or East. We think that this approach is the most efficient one taking into account the Q-table "threshold".

We have set the conditions to get the relative positions by creating a variable to see if there is a wall in each direction. Then, we know for example that if the north is not a legal direction, wallNorth will be True (there is a wall there). Same for the rest of the directions.

Finally, we have a loop that evaluates if the coordinates of the ghost position and Pac-Man position are larger or not to see the relative positions, considering for example that there is no way that Pacm-Man is at the East and West at the same time.

2 Reward function

To implement the reward function correctly we needed to first define an indicator of what we consider good behaviour.

In this particular setup, we need to program Pacman to eat all the ghosts.

When abstracting this problem to the maximum we find that this means that we need Pacman to increase its score (which implies it has effectively eaten a ghost), this means that the variable score is a correct metric to define the reward function.

The reward function is defined as follows, we need to compare the associated score to the state in which we find ourselves in and the next state. For this, we will assign two variables x and y to our state and nextState, and compare their signs to see whether there should be a reward or a penalization:
Disclaimer! All clases are excluding one another; if one is fulfilled, none of the rest will be.

1. The product of x and y is larger than zero and x is larger than zero (then, both x and y are). For this case, if x (current state) is larger than y (nextState), the reward equals 1. Else, there is a penalization of -1. The product of x and y is larger than zero and x is smaller than zero (and so is y, both values are negative). Then, if the absolute value of the current state x is larger than the absolute value of the nextState y, there is a reward. Else, there is a penalization.

2. If the product of x and y is negative (smaller than zero) and x is positive (then, y is negative), there is a penalization of -1 because the current state score will always be larger than the nextState score. environment.

3. Same applies the other way around, if the product is negative and x is the negative value, there will always be an improvement, so there will always be a reward.

4. If x is zero and y is larger than zero or x is smaller than zero and y is zero, there will always be a reward too.

5. If x is zero and y is smaller than zero and if x is larger than zero and y is zero, there is always a penalization.

We did not need the case of both x and y being zero, since there is no way the score does not change from a tick to another one. Our function returns the

corresponding reward or penalization, **keeping in mind that the penalization will be bigger or smaller depending on the distance to the closer ghost**.

3 computePosition function

This function must return a row value, so we try that the row is defined by a a number between 0 and 63, as

$$2^6 = 64 \tag{1}$$

possible values.

Our equation to return this value is

$$\sum_{n=1}^{6} 2^n \times V_n \tag{2}$$

where V refers to the counted variables from left to right, giving them a value 0 in case it is False and 1 if it is True. This way, our code is based on a sequential behaviour: whenever there is a change from True to False, the row will change too.

4 Failures from our code and how we solved them

Regarding the fixing of previous errors, we changed the Q-matrix in order to make it more feasible; in tutorial 4 we had very big values in the Q-table, so our agent was not running properly until we found out a mistake in the reinforcement learning equation.

Secondly, first we decided to use relative position to the closer ghost and the legal actions to configure our matrix, but it was not very accurate so we took into account the method gameState.data.ghostDistances, which takes into account

the distances but keeping in mind the walls.
This way, our Pac-Man got considerably better in the maps with walls.

Another change was made in the reward function later on: initially, Pac-Man gave rewards and penalizations equally, both 1 and -1.
With this code, Pac-Man was very penalized in bigger maps so at some point when many steps were done incorrectly (at the beginning, overall) we reached a very negative Q-table that did not allow Pac-Man to learn the movements. To fix this issue, with incorporated to the penalization system a new rule: if Pac-Man was moving in the right direction(to the closer ghost), the penalization would be smaller at each step.

This way, our Q-matrix was not so negative and there was a considerable change in the values when the agent eats a ghost (in our old version, as we reached very negative values, there was only one point of reward for many points of loss). Also the other way around: if the agent gets further away form the ghost objective, penalization will scale up.

5 Experiments

All our experiments can be summarized in the next questions:

1. Do score converge after some games to some number in all maps?
We have proved in all maps for different values of ε, keeping the discount and learning rate fixed, noticing that in all of them there is a convergence when we play many consecutive times. Moreover, we will prove it by adding a plot of both maps 1 and 2 (respectively), in which it is easy to see that the performance of Pac-Man in terms of score for all ε are similar.

2. In which map does Pac-Man perform better?
Consider the following tables of measures, with different ϵ and keeping α and the discount at 0.8, the values on it are the average score in 20 games:

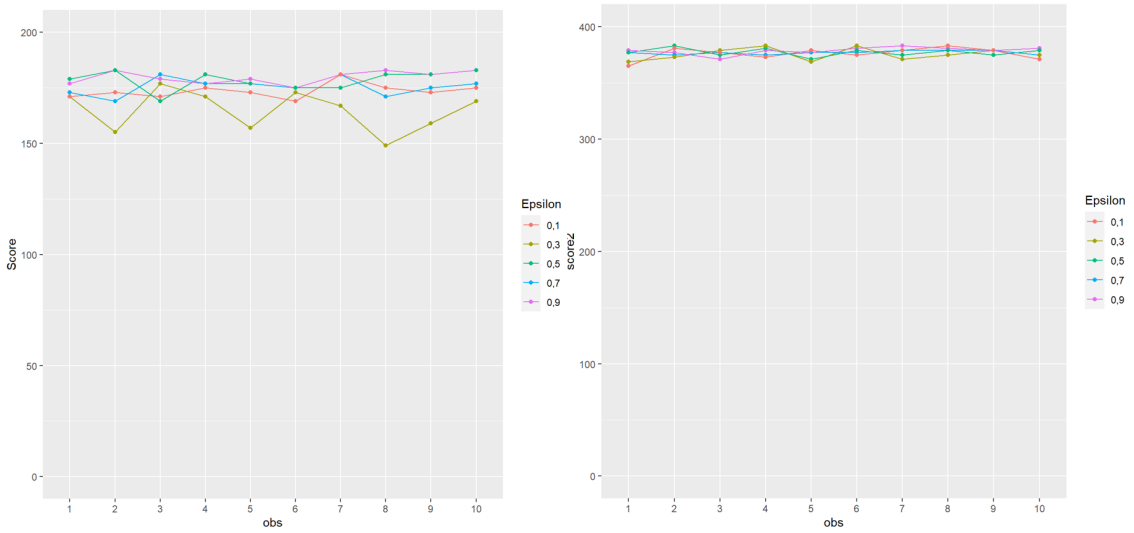

Table 1: Average scores depending on ε

	$\varepsilon = 0.1$	$\varepsilon = 0.3$	$\varepsilon = 0.5$	e = 0.7	e = 0.9
Map 1	170.8	179	**181.6**	179	180.8
Map 2	379.8	378.6	**380.5**	377.1	377.8
Map 3	542.6	521.2	524.4	**526**	512.3
Map 4	535.6	544.6	540.4	**546.6**	598.2
Map 5	759.6	638.8	723.2	**779.4**	711.6

Note: In Map 5, one more proof has been tried around $\varepsilon = 0.7$ (maximum value). With $\varepsilon = 0.8$, the score is even better (819.9).

If we consider that the best estimation is given by the points, Map 5 is the winner. Despite this, comparisons in performance cannot be driven uniquely by this, since the food in each map and the obstacles (walls), as well as the ghosts, differ at each case.
In terms of converging to a unique value, Map 2 is the one with smallest threshold (all the scores are close to each other).

3. Is there a relationship between ε and score?
From the previous tables, we may infer some clear tendency between the epsilon growth and rise in points, but we cannot affirm that it is linear nor being fulfilled in all cases.

Table 2: Average scores depending on α

	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 0.6$	$\alpha = 0.8$	$\alpha = 0.9$
$\varepsilon = 0.8$, $\gamma = 0.8$	751.6	727.6	757.2	**772.8**	708.2

Table 3: Choosing γ based on Map 5 best parameters'

	$\gamma = 0.2$	$\gamma = 0.5$	$\gamma = 0.9$
$\epsilon = \gamma = 0.8$	702.4	661	606.2

4. How does α (learning rate) affect the performance of Pac-Man?
The step size or learning rate α is a parameter $0 \leq \alpha \leq 1$. If we set the step size to 0, the agent learns nothing from the new actions. In the other extrema, the agent completely ignores prior knowledge and only values the most recent information.
In our case, the agent has been measured in Map 5 with the $\varepsilon = 0.8$ and $\gamma = 0.8$ (proven already that for this epsilon the score is the best possible) in order for us to know how this parameter behaves. The results are available in Table 2.

Note: One more proof has been tried in $\alpha = 0.8$ (maximum value), the score is even better (772.8).
Our agent works very well overall when it balanced between both conditions, inclined to ignoring the prior knowledge (for 0.7 or 0.8).

5. How does the discount affect the performance of Pac-Man?
Regarding the discount factor, it is the parameter that defines how important it will be the next reward. It has the same limits as the learning rate, $0 \leq \gamma \leq 1$. When it is set to zero, it completely ignores future rewards. When it is set to 1, it will look for higher rewards in the long term. What about our agent? We can get the insight that our Pac-Man is more interesting in present rewards rather than in th ¡e long term ones, based on Table 3.

6 Conclusions

To get a final conclusion from this assignment, it was very rewarding to see how our Pac-Man became more self-conscious when we took some measures in the code.

We have not found more big problems apart from the ones mentioned before, while in assignment 1 technical problems were an issue to deal with. The task of data mining itself (recopiling the data) has given us very good insights on how Reinforcement Learning works, what are the parameters to take into consideration and which is their importance for the process of learning.

Both of us feel like this has been so far the project in which we developed more skills, as well as we are satisfied with the approach we haven taken on it.

As a final brochure to the subject, we would like to mention that Machine Learning (as a subject, all in all) has given us more background on procedures and programming, as well as useful ways to fix some common problems we will need to face in the close future.