Università degli studi di
Milano-Bicocca

Decision Models

Final Project

Methodological Approaches for Multi-agent RL games

Authors:
Riccardo Cervero - 794126
Federico Moiraghi - 799735
Pranav Kasela - 846965

August 30, 2019

Abstract

1 Introduction

Reinforcement Learning is the branch of Machine Learning gathering several techniques designed to create a system, an agent, capable of learning the best way to achieve goals within an environment, whose elements - such as, for example, obstacles or objectives - can also vary over time. In other words, this agent simulate the process of human understanding, by exploring the initially unknown environment and consequently receiving rewards or penalization based on the goodness of the choices made.
In this study, we especially set ourselves the goals of:

1. Using different methodologies to perform the same Reinforcement Learning experiment (Game 1);

2. Deepening the dynamics among multiple separate agent, each of which will learn automatically and independently how to win the game by looking and reacting to other agents' choices. In particular, the two dynamics of "competition" and "cooperation" between agents will be examined.

To do so, four "worlds" have been built, with different rules and elements - such as fear or randomly moving obstacles -, within the agents will be trained over several epochs to maximize the reward, beat the rivals or help the allies in the best way possible.
The following sections will present games' and agents' characteristics.

2 Game 1: Dragon Hunting

This first experiment has been created with the aim of employing Q-Learning algorithm, Deep Q-Learning and Deep Q-Learning with the integration of Genetic Algorithm on a single system whose objective is beating a non-intelligent rival.
The environment the three agents will be trained within is defined as a class named "World" in the Python programming language, by which it is possible

to create the elements they interacts with during the learning process. First of all, it is initialized a Numpy array with dimensione 15x15 as an empty grid, to which various elements are added, so as to compose the world the agent explore:

- Obstacles, whose presence in the grid is denoted by the special character '▮', and which can be situated randomly - if no "entrance" location is specified-, or in a precise position over the epochs;
- A treasure (♔), which can be located in a randomly chosen coordinate or predefined too. It can not be moved, but only caught by the agent;
- A dragon (☠), which is the non-intelligent enemy of the agent. It can be spawn randomly, or, in case the position of treasure was preselected, situated close to it, in order to make the game more difficult. In this experiment, the dragon is not capable to learn any strategy to win the game over the epochs or react to agent's moves, but it merely moves randomly through the grid following the four directions 'up', 'down', 'right', 'left';
- *Bilbo* (☺), the agent to be trained, which is always generated at a random position. Unlike the other components, Bilbo moves around the world trying to maximize the total reward of his game and avoid penalties in best way possible, which is the core operating principle of any Reinforcement Learning methodology.

The creation of these components is carried out by the function *"random_spawn"* at locations where there are no obstacles or other characters, so that the grid remains consistent, while the function *"is_border"* checks whether a cell at a given coordinate is borderline.
Therefore, Bilbo's goal is to reach the cell where the treasure is without getting caught by the dragon, condition that occurs when the two end up in the same cell. The game ends when Bilbo is killed by the dragon in the terms just described, or when he manages to get the treasure, escaping from the enemy among the obstacles scattered around the world. In the meanwhile, another function returns the reward based on every game state:

- If the treasure has been caught by Bilbo, he gets a positive reward of 10;

- If Bilbo has been killed by the dragon, he gets a penalty of -5;
- At each moves, Bilbo receives a penalty of -1. This negative reward serves to incentivize Bilbo to find shorter routes to reach the treasure;
- If Bilbo dumps into an obstacle, so his position remains the same, he gets a penalty of -1, so that he learns to avoid them.

Reward values can be a negative constant or 0, but in order to obtain a faster convergence, it's better to give the agent a positive reward when he gets near the objective.
Another Python class named "Agent" serves to set the main characteristics of Bilbo:

- He can move in the same four direction of the dragon - 'up', 'down', 'right', 'left';
- He is affected by a "fear" factor, in order to simulate an aspect of real human exploring process. This is produced by a function which returns a value based on the distance from the dragon: if Bilbo is far from it, the "fear" value is close to 1; otherwise, as the distance from the dragon decrease, the value increases at most to 2. This measure is calculated as follows:

  $$\frac{d}{1+d}$$

  with d representing the Euclidean distance between the dragon's position (D) and Bilbo's location (B):

  $$d = \sqrt{(D_x - B_x)^2 + (D_y - B_y)^2}$$

  This will consist in a divisor of the ϵ parameter used in Reinforcement Learning algorithms to produce random moves in the grid and allow a better exploration of the world. As one can see from the formulation, this factor is a normalized distance, used to prevent ϵ to be too much affected by it.
  In conclusion, in this way, when Bilbo approaches the dragon, his fear level increases and the chance of a random exploration as well.

Given this general aspects of Game 1, three methodologies have been exploited to provide the agent with a learning process, each based on an agent generalization provided by a new Python class.

2.1 Q-Learning algorithm

The first approach consists in Q-Learning algorithm, which is based on the propagation of potential reward from the best possible actions in future states. Following this idea, each action in each state is related to a Q value, updated when new relevant informations about the environment, derived from exploration, are made available. The Q-Learning algorithm utilizes the following updating rule:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma\ max\ Q(s',a') - Q(s,a))$$

with $Q(s,a)$ Q value of current state and $Q(s',a')$ Q value of next state; r the reward obtained by taking action a at the state s; γ discounting factor regulating the impact of future rewards; α learning rate. Hence, the worth of an action in a certain state also depends on the discounted worth produced by the best action in the next state s', and on the extent of the steps to the convergence we selected with parameter *alpha*. In other words, this policy makes agent estimate the goodness of an action in a certain state based on the cascaded, discounted reward from the next states. In this way, at state s, Bilbo won't choose the action providing the best rewards only in the current state, but the action which leads to the best total rewards over all the states, while exploring the world in order to discover the rewards or penalties of all action. As Q-values are updated, they are stored in a Q matrix, so that Bilbo can learn from past experiences. This exploration process, that consists merely in random moves around the "gridworld", is fostered by ϵ parameter, which depends on "fear factor" as aforementioned. More over, ϵ decays by being multiplied to a "decay factor" at each step, up to a minimum of 0.01. In conclusion, this process can be explained as follows: if a number extracted from a uniform between 0 and 1 is less than ϵ or the Q values of all possible actions in the current state are null, the system performs a random step. Otherwise, it performs the action related to the maximum Q value in that state.
The values set for hyperparameters are the following:

- $\alpha = 0.5$
- $\gamma = 0.8$
- $\epsilon = 0.2$

- $decay\ factor = 0.99$

Then, the limits of epochs and episodes executable are both set to 1000. During the iterations, the performances of Bilbo are tracked by counting the number of wins, cases where he manages to stay alive and get the treasure, and losses, when he ends up killed by the dragon.

2.2 Deep Q-Learning approach

2.3 Genetic Algorithm applied to Deep Q-Learning algorithm

3 Game 2: Competition between two agents

4 Game 3: Competition among five agents

5 Game 4: Multi-agent Cooperation

6 Results and Evaluation

Descrizione dei risultati, performance, immagini.

7 Discussion

Deep Learning per il problema multi-agente ed altre possibili migliorie.

8 Conclusions