Delivering Research Outputs

Kevin Dhëmbi

I. Introduction

***Abstract*—This research paper consists of the required work done for the project defined by the assignment: Delivering Research Outputs. The paper is created for the Module: Artificial Intelligence, lectured by the lecturer Petr Švarný.**

THIS paper presents the results of the prototype, and draws a conclusion based on the outcomes of it. The prototype is designed to solve the problem of a car that is on a one-dimensional track, positioned between two mountains. The goal is to drive up the mountain, however, the car's engine is not powerful enough to drive up itself. Therefore, the car must gain momentum by driving backwards. The paper gives clear evidence of the prototype's abilities to solve the game, using Reinforcement Learning.

II. The Mountain Car Problem

The problem, as mentioned above, involves a car that needs to climb up the mountain, in order to complete the game. It will be solved using OpenAI Gym. The car, further called "the agent", starts the game on the bottom of a valley with an initial speed of zero. Then, the agent may choose to accelerate to the left, right or halt any acceleration. For the agent to learn how to solve the game continuously, rewards must be introduced to him. Rewards are part of reinforcement learning, which will be presented shortly. The initial score of the agent is zero. If the agent catches the flag, hence reaches the top of the mountain, a reward of zero will be awarded to him. On the contrary, if the agent fails, a reward of minus one will be awarded. After numerous attempts, the agent should be able to have stable rewards by the end of the simulation.

III. Reinforcement Learning

Reinforcement Learning is a Machine Learning training method based on rewarding desired behaviors and/or punishing undesired ones. In order for a system to be called a Reinforcement Learning system, it must have an agent and an environment. The agent acts with actions upon the environment and the environment provides the agent with a reward and a new state. Furthermore, the basic parts of a Reinforcement Learning system are displayed on the table below [Table 1].

TABLE I
Basic Parts of a Reinforcement Learning system

Environment	The world in which the agent operates.
State	The current situation of the agent.
Reward	The feedback from the environment.
Policy	Method to map agent's state to actions.
Value	Future reward that an agent would receive by taking an action in a particular state

IV. Q-Learning

Q-Learning is a Reinforcement Learning technique used for learning the optimal policy. The letter "Q" stands for quality. Its goal is to find the optimal policy by learning the optimal Q values for each state-action pair. An optimal policy is better than, or at least the same as all other policies. To be specific see [Fig.1.].

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in \boldsymbol{S}.$$

Fig. 1. Optimal Policy

An optimal policy has an associated optimal state-function [Fig. 2.]. V star gives the largest expected return achievable by any policy π for each state.

$$v_*(s) = \max_\pi v_\pi(s)$$

Fig. 2. Optimal State-Value

In addition, an optimal policy has an optimal action-value function, displayed in [Fig. 3.]. Q star gives the largest expected return achievable by any policy π for each possible state-action pair.

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

Fig. 3. Optimal Action-Value

Q-Table is a look up table where the maximum expected future rewards for action at each state is calculated. Initially, it will be filled with zeros, hence it won't know anything. The Q-Learning algorithm updates the Q values in an iterative sequence for each state-action pair, until the Q function converges to the optimal Q function, which is called Q*. The following equation is used for this algorithm [1]:

$$Q[s,a] = Q[s,a] + alpha * (R + gamma * Max[Q(s',A)] - Q[s,a]) \quad (1)$$

Where;

Alpha is the learning rate. It can be set between zero and one. If 0, the values on the table will never be updated, hence the agent can never learn. Setting a high value, such

as zero point nine, can make the agent learn quicker, but the learning would be inefficient.
Gamma is the discount factor. It quantifies how much importance we give for future rewards. If gamma is closer to zero, the agent will tend to consider only immediate rewards. On the contrary, if it is closer to one, it will pay much more attention to the future rewards.
Max[Q(s', A)] gives a maximum value of Q for all possible actions in the next state.

V. Exploring and Exploiting

Initially, all of the values in the Q-table are set to zero. Therefore, the agent, in this case the car, does not know which action to start with. By exploring the environment the agent will only take random actions, while updating the Q-Table. However, this alone would not bring to the completion of the game. The agent must find a way how to efficiently exploit the previous states.

Essentially, the agent needs to have a balance between exploring and exploiting the environment. This balance will make the agent smart enough to regularly complete the game, and at the same time exploring the environment in order to optimize the solution. To reach this balance, the Epsilon-Greedy strategy, where epsilon refers to the probability of choosing to explore, must be introduced to the agent. Without this strategy and with a preset epsilon value lower than one, it would require much more time for the agent to learn how to continuously solve the game.

VI. Solving The Game

The technique used to solve the Mountain Car problem is discretization. This approach takes the continuous state variables, in this case: position and velocity, and turns them into discrete states. These continuous state variables are bucketed into multiple discrete states, respectively twelve and twenty buckets. A bucket is often called a generated sample. However, a notorious disadvantage of this approach is that information gathered from one state is not used to evaluate another state, hence a big number of games played by the agent is required to learn how to beat the environment.

The number of games that will be played is initially set to fifty thousand, alpha is set to point one, gamma is set at point ninety-nine and epsilon is set to one. The maximum number of steps that the agent can take per episode is a thousand. The actions that the agent can take are three, accelerate left, right or do nothing. On the other hand, the amount of states that the agent could be in, would be the product of the samples generated of position and velocity, approximately two hundred seventy-three. As mentioned before, a Q-Table must be initialized, so that the values that the agent will collect while exploring, will be stored into. In order to return a state an observation of the environment must be taken. An observation is an environment-specific object representing your observation of the environment. The game is solved through repetitively using the Q-Learning algorithm, that was introduced earlier, while using the epsilon greedy strategy, until terminated.

Initially, epsilon will be set to one. This will allow the agent to only explore the environment. At the start of each try, further called episode, the epsilon will decay linearly, as described in the code. Eventually, the agent will exploit the environment, once it had the opportunity to explore and learn more about it. To determine what the agent will choose, a random number is generated. If it is smaller than the epsilon, the agent will choose exploration, which is randomly choosing an action and exploring what will happen in the environment. Otherwise, the agent will choose the action with the highest Q-Value in the table, hence exploiting the environment.

VII. The Outcome

After fifty thousand games, the agent shows clear evidence of learning. See [Fig. 4.].

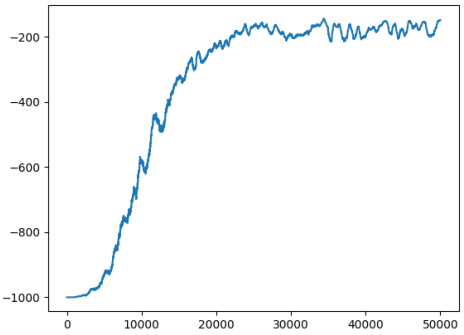

Fig. 4. Reward per Thousand Games

In the first thousands of games, the epsilon number is still high, hence the agent is still exploring the environment, and not actively trying to solve the game. Nevertheless, while the epsilon decays with time, the agent significantly increases the score, therefore achieves to solve the game in less moves. However, towards the end the score is not stable. This happens because the state-action space is relatively high. If a bigger number of games is played, the curve will be much more stable. In addition, while the agent plays the game more, he accomplishes to solve it more quicker than in the beginning, when he had to take the maximum steps per episode to just explore the environment.

VIII. Conclusion

In conclusion, by the end of the simulation, the agent performed well. It has learned how to solve the Mountain Car problem, via Reinforcement Learning. As mentioned before, allowing it to perform more games, will bring a better and more stable result.

REFERENCES

[1] R. S. Sutton and A. G. Barto. *Reinforcement Learning An Introduction*. MIT Press Ltd., 1998.