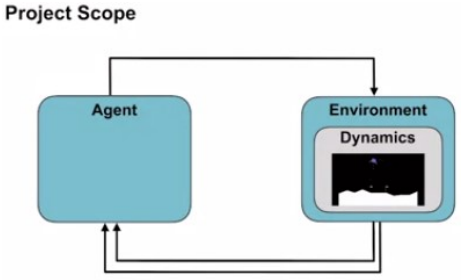
Realize a Reinforcement Learning System

Andrea Rossolini

March 2021

1 Introduction

The goal of the project is to land a lunar module on a specific spot on the moon.The Landing zone is always in the same location, but the shape of the ground around it may change.

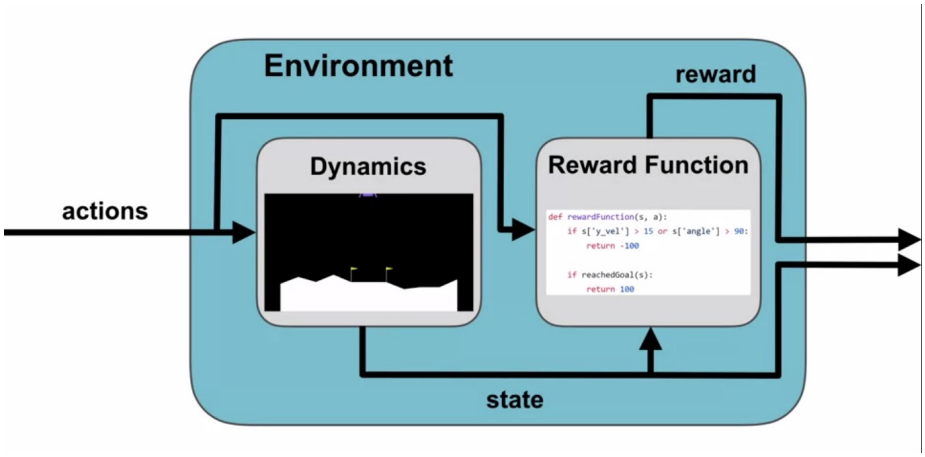


1.1 Agent

From the lunar module we can fire the main thruster or either of the side thrusters to orient the module and slow its descent. The state is composed of eight variables. It includes the XY position and velocity the module as well as its angle and angular velocity with respect to the ground. We also have a sensor for each leg that determines if it is touching the ground.

1.2 Environment and States

The environment inputs the action that actually is given to the dynamics function along with the current state to produce a next state. The next state in action will then be passed to reward function that encodes the desired behavior. Finally, the environment will omit the next state and the reward.



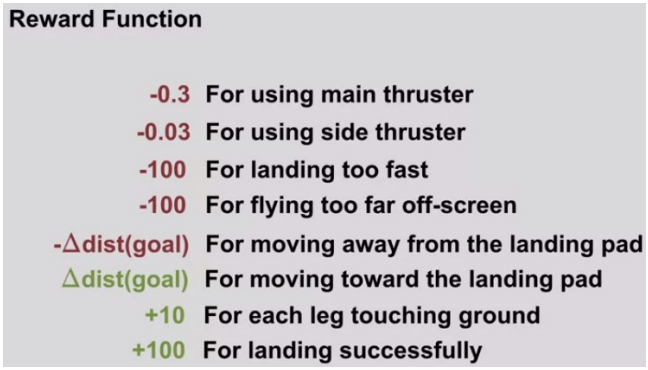
1.3 Actions

There are four actions that the agent can take, the agent can fire the main thruster, fire the left thruster, fire the right thruster or do nothing at all on this time step.

- Fire main thruster
- Fire left thruster
- Fire right thruster
- Do nothing

1.4 Reward Function

You will need to implement the reward function for this environment. Fuel is expensive and the main thruster uses a lot of it. We want to discourage the agent from using the main thruster more than necessary. The side thrusters use less fuel, so it is less bad for the agent to use those frequently. We want to encourage the agent to move towards the goal. So it will lose some reward based on how far it moved from the goal since the last time step. Let's try to discourage the agent from learning to pile the module to the surface in ways that might damage the equipment and will also discourage flying off into outer space or a distant creator never to be seen again.



The agent will be rewarded for each leg that it manages to get touching the ground. And the agent will receive a large reward for successfully landing in the landing pad at an appropriate velocity.

2 Choosing the Learning Algorithm

Following the "Algorithm Map" in the resources we can argue as follows:
First step, *can we represent the value function using only a table?* Let's recall the state space of the lunar lander problem. The agent observes the position, orientation, velocity and contact sensors of the lunar module. Six of the eight state variables are continuous, which means that **we cannot represent them with a table. And in any case we'd like to take advantage of generalization to learn faster.**
Next ask yourself, *would this be well formulated as an average word problem?* Let think about the dynamics of this problem. The lunar module starts in low orbit and descends until it comes to rest on the surface of the moon. This process then repeats with each new attempt at landing beginning independently of how the previous one ended. This is exactly the definition of an **episodic task**. We use the average reward formulation for continuing tasks, so that is not the best choice here. So let's eliminate that branch of algorithms. Next we want to think about *if it's possible and beneficial to update the policy and value function on every time step*, we can use Monte Carlo or TD. But think about landing your module on the moon. If any of our sensors becomes damaged during the episode, **we want to be able to update the policy before the end of the episode**. We expect the TD method to do better in this kind of problem. Finally, we want to learn a safe and a robust policy in our simulator so that we can use it on the moon. *We want to learn a policy that maximizes reward*, and so this is a **control task**. This leaves us with three algorithms, SARSA, expected SARSA and Q-learning. Since we are using function approximation, learning and epsilon soft policy will be more robust than learning a deterministic policy (deterministic policy is a suboptimal in this example). Expected SARSA and SARSA, both allow us to learn an optimal epsilon soft policy, but Q-learning does not. Now we need to choose between expected SARSA and SARSA. We

mentioned in an earlier video that expected SARSA usually performs better than SARSA. So, let's eliminate SARSA. **Expected SARSA** is our choice.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \sum_{a'} \pi(a'|S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t))$$