Deep Q Learning for Rocket Landing

Ibis Prevedello

March 2018

1 Introduction

The goal of this project is to study and implement a Reinforcement Learning algorithm in order to solve several games from the Gym Toolkit [3]. The algorithm chosen is called Deep Q Learning, and uses a Deep Neural Network to learn the Q Function. Besides the implementation, also some improvements are tried with Double Deep Q Learning and Prioritized Experience Replay in order to improve performance.

2 Description of the problem

Besides the idea is to develop an algorithm that can be used for more than one game, this report will contain only the solution for the environment called 'Rocket Landing', which is inspired on SpaceX idea of reusable rockets.

The rocket starts from a high altitude, out of the screen, as shown in image 1, and the goal is to control it using the trusters in order to land it on a drone on the sea. The landing also needs to occur smootly in order to avoid demaging the rocket legs.

3 Formal model of the problem

To control the rocket it is necessary to give to the learning algorithm the actual state of the problem, called state variables and take one of the actions from the control inputs. After every iteration, which is a frame of the simulation, an action will be calculated and a result of that action will be returned to the algorithm as a reward and the result state.

3.1 State variables

For the state variables, it is possible to limit the number of variables used for the training, however it was decided to use all the data available in order to have a better control of the rocket.

- x position

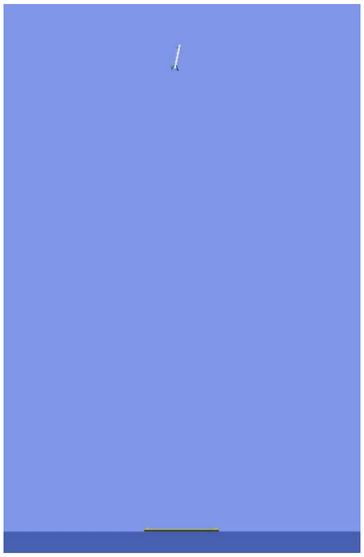

Figure 1: Game screenshot

- y position
- angle
- first leg ground contact indicator
- second leg ground contact indicator
- throttle
- engine gimbal
- x velocity
- y velocity
- angular velocity

3.2 Control inputs

Using this simulator it is possible to chose one of the two types of controls, or discrete or continuous. It was decided to use discrete control because the same code could be used to a bigger variety of other simulators inside Gym.

- gimbal left
- gimbal right
- throttle up

- throttle down
- use first control thruster
- use second control thruster
- no action

3.3 Reward function

The reward function for this environment envolves a series of variables, such as rocket's speed, distance to the goal, angle, angular velocity, fuel, ground contact and some other variables. It is only interesting to cite that the reward value improves as the rocket gets closer to the goal.

4 Solution algorithm

A algorithm well known is the Q Learning, which has the goal of finding a sequence of actions from some state that leads to a reward. Some actions may lead to a greater reward, others may achieve the goal faster.

The original Q Learning algorithm uses a table $Q(s, a)$, composed of rows of states and columns of actions. It uses the Bellman equation, which states that the expected long-term reward for a given action is equal to the immediate reward from the current action combined with the expected reward from the best future action taken at the following state.

$$Q(s, a) = r_0 + \gamma(r_1 + \gamma r_2 + \gamma^2 r_3 + ...) = r_0 + \gamma max_a Q(s', a)$$

In the case of this problem, because the states are constante, it is not possible to use a table, and it is substituted by a neural network. So the goal is to update the weights of the neural network, but it follows the same equation and the same process.

4.1 Hyperparameters

The hyperparameters used for this project are presented in the table below:

Parameter	**Value**
γ (reward discount factor)	0.99
learning rate	0.00025
target update (episodes)	1000
ϵ start	1.0
ϵ end	0.01
ϵ decay	0.0001
batch size	128
memory capacity	200000

4.2 Exploration and exploitation

The way the algorithm chooses the action is called policy. A variable ϵ is used in order to help with this decision, at the beggining it is important to chose random actions in order to get a better felling about exactly what is the response of the system given an action, it is called exploration. As the iterations passes this ϵ decreases in order to fine tune the parameters, what is called exploitation.

4.3 Experience replay

It was achnowledged by the authors of the original paper that using only a neural network to represent the Q function make it unstable [1].

When using online learning from each simulation step the problem is that the samples feed to the neural network are higly correlated, so the network will most likely overfit and fail to generalize properly.

The solution found is to store the last n experiences in a memory and instead of training the network with only one sample use a random batch from this memory, as described in [2].

4.4 Double deep learning

4.5 Prioritize experience replay

5 Implementation

The algorithm here described is divided in tree different parts, the memory, the brain and the agent. Each one has its own well defined role and it is important to keep the code separated and organized.

5.1 Memory

The memory is necessary to keep in memory a certain number of last actions executed by the agent. It is necessary to store the experience that will be used for Experience Replay.

The Experience Replay is a stratagy used to train the Neural Network not only based on the last experience, but on a batch with multiple experiences saved in the memory. For this part there are multiple ways to select this samples from the memory, random samples are one, and other more efficient algorithm is to determine an error for each sample and select those with a hier error.

Each experience saved to the memory have the following information:

- current state
- action
- reward
- next state

5.2 Brain

The Brain class encapsulates the Neural Network. It was defined with 2 hidden layers with 256 neurons each and ReLU activation function. The input number of neurons is the number of states and the output number of neuros is the number of actions.

This network is trained in order to aproximate the Q function and the target model is a simple copy of the model, however it is updated sporadically, after 1000 steps.

For the loss it is using the Huber loss function, it is a loss function used in robust regression, that is less sensitive to outliers in data than the squared error loss.

5.3 Agent

The Agent class is a container that encapsulates all the methods for the algorithm to run, such as the act, observe, replay and epsilon decrement.

Also some more higher level tasks are also implemented like save and load model, which enables the script to stop training and continue it from where it stopped.

6 Results

The first data that is useful to analize is the evolution of rewards over the training. In image 2 is plotted this evolution and it is possible to see the the reward is improving over time, even though

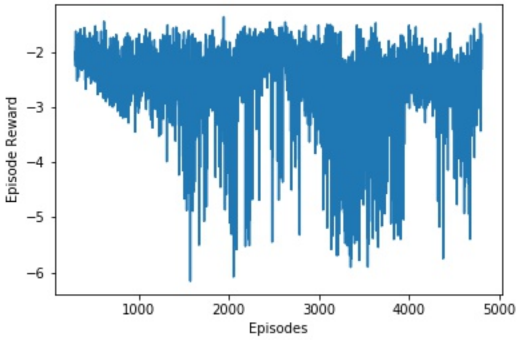

Figure 2: Reward over episodes

7 Conclusions

Another option

References

[1] Mnih et al. *Human-level control through deep reinforcement learning.* Nature 518, 2015.

[2] Lin L. *Reinforcement Learning for Robots Using Neural Networks.* PhD. thesis, Carnegie Mellon University Pittsburgh, 1993.

[3] OpenAI Gym.
`https://github.com/openai/gym`