Machine Learning Project

Fabio Massimo Ercoli

July 2024

1 Introduction

We're presenting two prossible Q-learning implementation approaches:

- Tabular based. Implemented using a numpy dictonary.
- Neural network based. Also known as Deep Queue Network (DQN).

2 Tabular Q-Learning

2.1 Running the project

To trigger the learning process run from the *machine-learning-project* directory the following code:

```
python qlearn-app.py
```

You should see a message similar to the following:

```
observation space Discrete(500)
action space Discrete(6)
avg return before learning -92.98608555399927
avg return after learning 2.5552244875845997
```

The observation and the action space description is related to the *Gymnasium* environment we used to train and test our agent, that is the *Taxi-v3 environment*[1] that has 500 discrete possible states and 6 possible actions.

The *avg return* is the average of the score gained rolling out a series of episodes. In the case of this project 5 episodes are executed before the training and 20 after the training. The expectation is that the agent can get a better score after we properly trained it.

An episod starts from the initial state and ends in case of a termination state is reached or is trunkated since the **max episode steps** value is reached.

This is the first value we can change to tune the lerning, in general this value should be great enough to allow the agent to learn and to rollout correctly the

[1] $https://gymnasium.farama.org/environments/toy_text/taxi$

strategy it learned. We decide to increase it from 200, that is the default, to 500 and this change seems to provided better performance to the learning.

How can we evaluate the goodneed of the learning? In this case we simpy compare the average score for an episode perfomed using a random strategy (before to learn) with the average score obtanied after the learn.

2.2 The rollout and the score

The score is a crucial concept, since the learning activity is entirly oriented to maximaize this value.

The score is the summation of all the reward we get from the environment every time we execute an action (notice that can be negative or positive), multiplied by the **discount factor** γ.

The meaning of this value is to promote not only the rewards collected, but also the speed with which we get them. This is another value that we need to balance, since if it is too low, we can penalize the learning of long term goals.

The score depends on the actions the agent chooses, those are randomly selected before the training and after the training they are chosen according to the learned policy.

The rollout procedure will collect the score for each epoch, applying the current discount factor to the reward. The score is averaged and retured to the caller.

2.3 The learning and the Q function

The output of the learning is a policy to choose for any given state (observation)[2] an action among all possible actions to apply. The Q function associate to a given state and a given action the expected total score of taking the action in the state and then continue to choose optimally the futher actions. So we can use the Q function to choose the policy as the action that maximaizes the expected total score.

We call it the *greedy policy*, since it maximaizes the expected score not conisidering the fact that unexplored paths may possibly lead to even greater scores, improving the Q function. At the begin of the learning we want always to apply a random strategy to learn as much as possible ($\epsilon = 0$). At the end of the learning on the other hand we want to exploit the knowledge we've acquired to perfectionate the policies on areas that we already know to be good ($\epsilon \approx 1$). In this project we use a linear decay from 0 to 1 of the **greedy factor** ϵ for the learning. Other decay functions of course are possible.

How should the learning last for? In this project the learning finishes as soon as we run a number of actions equals to the **learning steps**. In this project we set this value to 200,000.

[2]In the context of this project that state is always fully observable, so we will use the term state and observation interchangebly

The last crucial setting we present in this chapter is the **learning rate** α_0. The Q function is updated for supporting the indeterministic environments according to the formula:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max(Q(s',a'))] \tag{1}$$

Where s' is the state we get from s applying the action a, and the $\max(Q(s',a'))$ is calculated among all the possible action a' executable from s'.

2.4 The tabular approach

In this implementation the values for the Q function are stored as table items, in particular in order to represent only the subset of state we're interested in we use in this project a dictonary (instead of an array).

This means that every time we update an entry on the Q function we operate on a discrete value of the observation and on a discrete value of the action.

Thus in order to support with this approach continuous environment, we need first to apply a discretize of the state (and/or the action).

3 Deep Queue Network