Tópicos especiais em Inteligência Computacional II: Aprendizado por Reforço
Exercício 3: Deep Q-learning

Wouter Caarls
wouter@ele.puc-rio.br

17 de abril de 2020

A large breakthrough in reinforcement learning happened in 2014, with the introduction of Deep Q-learning[1]. It expanded RL into learning directly from visual information by using a deep neural network as Q approximator. In this exercise we will use the same algorithm, but for simpler tasks such as to limit the required learning time.

Start by installing the prerequisites by opening the Anaconda Prompt and running

```
(base) C:\Users\You> conda install swig
(base) C:\Users\You> pip install tensorflow gym[box2d]
```

1 Understanding the code

As oppopsed to the previous exercise, here you will implement both the simulation loop and the learning algorithm. Because deep neural networks are outside the scope of this course, the approximator is given.

The Python code for this exercise (`ex3.py`) contains four classes, see Table 1.

Tabela 1: Main classes

`ex3.DQN`	Deep Q network.
`ex3.Memory`	Replay memory.
`ex3.Pendulum`	Pendulum swing-up environment.
`ex3.Lander`	Lunar lander environment.

[1] `https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf`

Exercise 1.1 What would be the network structure for the `Pendulum` environment? How many inputs does the network need? By default, how many hidden layers does the network have and how many neurons are in each one?

Exercise 1.2 Create a `Pendulum` environment, and write a simple simulation loop of 1000 timesteps. Before the loop, `reset` the environment, and in every iteration `step` the environment with a random action taking from among its `actions`. `render` the environment after every step to visualize the system. After the loop finishes, `close` the visualization. What are the rewards returned by the environment?

2 Learning to swing up

You will now learn the pendulum swing-up problem (similar to the one you solved with SARSA) using DQN (see Figure 1).

Exercise 2.1 Create a basic learning structure, by enclosing your single-episode loop from the previous exercise within another loop. Before the outer loop, create the `Pendulum` environment, `DQN` network with the right number of state and action dimensions for the pendulum, and replay `Memory`. Add all experienced transitions `(prev_obs, action, reward, obs, done)` to the replay memory. `break` from the inner loop if `done` is `True`.

Exercise 2.2 Select actions according to the epsilon-greedy strategy. Note that you can evaluate the network for all actions in a certain state by calling `dqn(observation, env.actions)`. After you find the action index `i` with the highest value, use `env.actions[i]` to get the actual action to apply (similar to `take_action` in the SARSA code). Start with exploration rate 1, and multiply it by 0.97 after every episode while keeping it above 0.05. What is the effect of such an *exploration schedule*?

Exercise 2.3 Implement the Deep Q-learning learning step by sampling a minibatch of size 256 from the replay memory, calculating the target value for each transition, and training the network with the targets (use $\gamma = 0.99$). Do this only when the replay memory has at least 1000 transitions. Note that when calculating the target

$$y = r + \gamma \max_{a'} Q(s', a') \tag{1}$$

the $\gamma \max_{a'} Q(s', a')$ should only be added if the state is not *terminal*. That is, if `done = False` for that transition. Why is this the case?

Exercise 2.4 Reduce the rendering (maybe do it only once every 10 episodes), and run the algorithm, printing the total reward per episode. It should converge to values above -300 in around 50 episodes. Plot the value function using `env.plot(dqn)`. Is it what you would expect? Why (not)?

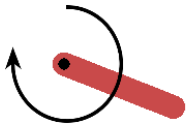

Figura 1: OpenAI Gym Pendulum-v0 environment

Exercise 2.5 Reduce the hidden layer sizes to `[5, 5]` and run the algorithm again. What changes in the output of the plot? Explain.

3 Learning to land

Now we move to a more interesting problem: landing on the moon (see Figure 2)! The greatest difference is the number of state dimensions, which would be prohibitively large to represent with a Q table.

Exercise 3.1 Change the environment to `Lander`, fix any runtime errors and run the algorithm again. What is the result?

Exercise 3.2 Implement a *target network* by creating a second identical `DQN` object and copying the original network's weights to it every 50 timesteps. Use `target <<= dqn` to copy the weights. Use the target network to

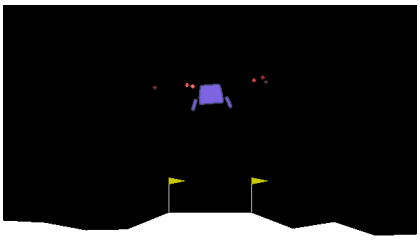

Figura 2: OpenAI Gym LunarLander-v2 environment

calculate the target values in Eq 1.The learning should converge to values above 100 in around 200 episodes.

Exercise 3.3 Try to find hyperparameters (discount rate, exploration schedule, batch size, target network update interval, hidden layer parameters) that give more consistent performance. Document your search. Note that the results will be stochastic!

Send your report in pre-executed ipynb format to `wouter@ele.puc-rio.br`.