NNDL Homework 3:
Deep Reinforcement Learning

Alessandro Lovo

February 10, 2021

1 Introduction

This homework will deal with Reinforcement learning using *OpenAI*'s *gym* environments. In particular we will start with the simple *CartPole*, where the goal for the agent is to balance a pole on a cart, first providing the agent with high level knowledge of the environment, then giving it just the screen pixels. Then another gym environment will be tested. In every case the goal is to consistently beat the games with as few training episodes as possible.

1.1 General framework

The interaction between the environment and the agent and learning of the latter rely on the following framework of python classes and functions:

- **Policy net**: different architectures for deep neural networks taking as input the state of the environment and providing as output the Q-values for each possible actions.
- **Policy**: function for choosing which action to take given the Q-values and a generic parameter β. The possible policies are: *random* (choose action randomly, ignoring the Q-values), *greedy* (choose the action with the highest Q-value), ϵ*-greedy* (with probability ϵ choose a non optimal action, otherwise the greedy one; here β is ϵ), *softmax* (here β is the temperature parameter and the probability of choosing action a is proportional to $e^{q_a/\beta}$). Both ϵ*-greedy* and *softmax* policies reduce to the *greedy* one whe $\beta = 0$.
- **Agent**: class for handling the interaction with the environment: it observes the state of the environment, passes it to the *policy net* obtaining the Q-values, which uses, given a *policy* to choose the action to take and pass back to the environment, receiving a reward and the next state.
- **Replay memory**: class containing a list of finite capacity with tuples [state, action, next state, reward] from which we randomly sample when training the agent. Once we reach the maximum capacity new data will overwrite the oldest ones.
- **Exploration profile**: class for scheduling the value of the β parameter for the policy of the agent at every episode of training. Said value can either be predetermined, i.e. an exponential decay with the episode number, or depend on the current performance of the agent.
- **Evolver**: class for handling the training of the *policy net* of the *Agent*. It optimizes the smooth L1 loss between $policy\_net(state, action)$ and $reward + \gamma \cdot target\_net(next\,state, best\,action)$ where γ is the discount rate and the target net is a periodic checkpoint of the policy net. Data [state, action, next state, reward] are sampled from the *replay memory* and the best action is computed using the greedy policy on the target net.

2 Cart-Pole

This environment consist of a cart able to move on a 1d rail with a pole attached, initially facing upwards. The state consists of four real numbers: cart position (x), cart velocity (v), pole angle (θ), pole angular velocity (ω) and the agent has two available actions: push the cart to the right or to the left. The goal is to keep the pole from falling (angle less then 12 degrees) and the cart on screen for 500 steps.

In the vanilla version at every step the agent receives a reward $r = 1$ until the pole falls, so the agent struggles to understand the consequence of its actions and learning is pretty slow, so I proceeded in modifying the reward as $r = 1 + w_x|x|^{e_x} + w_\theta|\theta|^{e_\theta}$

3 Cart-Pole with screen pixels

4 Other gym env