RL211 - HW4

Omri Himelbrand - 200863843
Nitzan Cohen - 203980750

January 7, 2021

Contents

1 Introduction

The code uses Actor-Critic algorithm to approximate the value function $\hat{v}(s, W)$ and the policy $\pi(a|s, \theta)$.
Using LFA for approximating $v(s)$ with our set of weights W along with extracted feature-vector representing the state and $\pi(a|s)$ with our set of of weights θ for each action given the state along with soft-max.
Denote α^W, α^θ as learning rates for updating W, θ respectively, the code implements Auto-Step-Size Selection method for α^W and α^θ separately, which theoretically proposed by us and empirically tested to work and converge.
Denote $v_{best}(0)$ as the highest $v(0)$ value so far through the algorithm run, after each run of Actor-Critic algorithm and evaluation of $v(0)$, we calculate the $ratio = v(0)/v_{best}(0)$ and multiply α^W, α^θ using $ratio^2$, $ratio^3$, respectively, upper bounded by the initial α^W, α^θ, though implementing Auto-Step-Size Selection.

2 Running the solution

2.1 Running as a script

```
usage: hw4.py [-h] [-human] [-gamma G] [-d] [-ms MAX_STEPS] [-es EVAL_STEPS]
              [-png PNG_SUFFIX] [-relax]

Actor - Critic for AI-Gym Mountain Car.

optional arguments:
  -h, --help        show this help message and exit
  -human            use this flag to run human agent
  -gamma G          a float for gamma in [0,1] (default: 1.0).
  -d                use this flag to get debug prints
  -ms MAX_STEPS     a int for number of maximum steps for learning.
  -es EVAL_STEPS    a int for number of steps between evaluations.
  -png PNG_SUFFIX   a suffix for png out file
  -relax            use this flag to use 500 steps episodes
```

2.2 Running as module

```
import hw4

hw4.main()
```

3 Code details

3.1 Global variables

DEBUG: A boolean which is initialized to False.
MAX_STEPS: An integer to indicate the number of steps for the entire learning process, initialized to 150000.
EVAL_STEPS: An integer to indicate the number of steps between running evaluations simulations, initialized to 500.
RELAXED: A boolean value, indicating whether or not to run the relaxed version (500 steps limit per episode).
P_CENTERS: A list containing the centers of the positions' Gaussians.
V_CENTERS: A list containing the centers of the velocities' Gaussians.
CENTER_PRODUCTS: $P\_CENTERS \times V\_CENTERS$.
SIGMA_P: variance/std of position.
SIGMA_V: variance/std of velocity.
COV: the co-variance matrix.
INV_COV: COV^{-1}
P_I: float value of interval size for centers of position.
V_I: float value of interval size for centers of velocity.

```
#initializing the last few globals can be done using these functions
def init_intervals(Ip=0.18,Iv=0.014):
    global P_I,V_I
    P_I = Ip
    V_I = Iv

def init_covariance(sigma_p=0.04,sigma_v=0.0004):
    global SIGMA_P,SIGMA_V,COV,INV_COV
    SIGMA_P = sigma_p
    SIGMA_V = sigma_v
    COV = np.diag([SIGMA_P,SIGMA_V])
    INV_COV = np.linalg.inv(COV)

def init_centers(p_half=4,v_half=4):
    global P_CENTERS,V_CENTERS,CENTER_PRODUCTS
    P_CENTERS = [(i + 0.5)*P_I for i in range(-p_half,p_half)]
    V_CENTERS = [(i)*V_I for i in range(-v_half,v_half)]
    CENTER_PRODUCTS = np.array(list(product(P_CENTERS,V_CENTERS)))
```

3.2 The functions

```
def init_env(max_steps=200):
```

Calls gym.make, sets the max episode steps for the created environment and returns it.

```
def set_debug(value):
```

Sets the global variable DEBUG to value.

```
def set_max_steps(value):
```

Sets the global variable MAX_STEPS to value.

```
def human_agent():
```

Prompts user to pick action for the next step of simulation, used for mostly for debugging.
Return value is a action (int). Note that in order to use this you must have "readchar" installed.

```
def evaluate(env,w,gamma,episodes_num=100,show=False):
```

Given the current weights (w), and the rest of the arguments seen in the signature, evaluates v_0^π, using a MC-like evaluation.
This function returns value v_0^π.

```
def apply_policy(Qhat,actions,eps=0):
```

This function apply the policy defined by the weights using ϵ-greedy scheme on $\hat{Q}$ (in case eps¿0), generating random number, if it is less than ϵ uniform random choice of action from action space, else argmax.

```
def AC(env, w, theta, gamma, alphas, actions, eps, max_step=5000, iters=0,\
                                        epsilon_decay=0.999, min_eps=0.1):
```

This function does Actor-Critic with LFA, as described in the introduction section.

```
def run_simulation(env,theta=None,human=False,show=True):
```

resets env to initial state, then runs simulation either using the given policy (by weights) or using a human agent.

```
def centers_distance(s:np.ndarray):
```

given a state returns vector of differences from the CENTER_PRODUCTS vector.

```
def state_features(x:np.ndarray):
```

calls *centers_distance*(x) and uses the returned value to computes the features of the state.

```
def init_weights(nA=3,seed=27021990):
```

Initializes the weights randomly using a seed.

```
def piApproximation(theta:np.ndarray,s:np.ndarray):
```

given θ and s, computes $\pi(\cdot|s,\theta)$.

```
def VApproximation(s: np.ndarray, w: np.ndarray):
```

given w and s, computes $\hat{V}(s,W)$.

```
def learn_policy(env, actions, gamma):
```

This function runs the whole learning process, running AC for EVAL_STEPS, then running evaluation.

After each evaluation the V_{init}^π is saved with the number of total steps taken so far, and we keep track of the best θ according to the evaluations so far, keeping it for return.

The return value of this function is: x - array of step counts, y - array of V_{init}^π collected, and the best θ.

```
def main(gamma=1,human=False):
```

This function is called when running the code as a script, but can be used as seen above, this function does the following:

- calls "init_env","init_covariance","init_intervals", and "init_centers"
- if human flag is set, runs a single simulation with a human agent.
- else calls "learn_policy", after running a simulation using a previous learned weights (if such exists).
- calls "run_simulation" using the returned θ
- After running the learning process and collecting all the x and y values, calls plot_results function

4 Plot

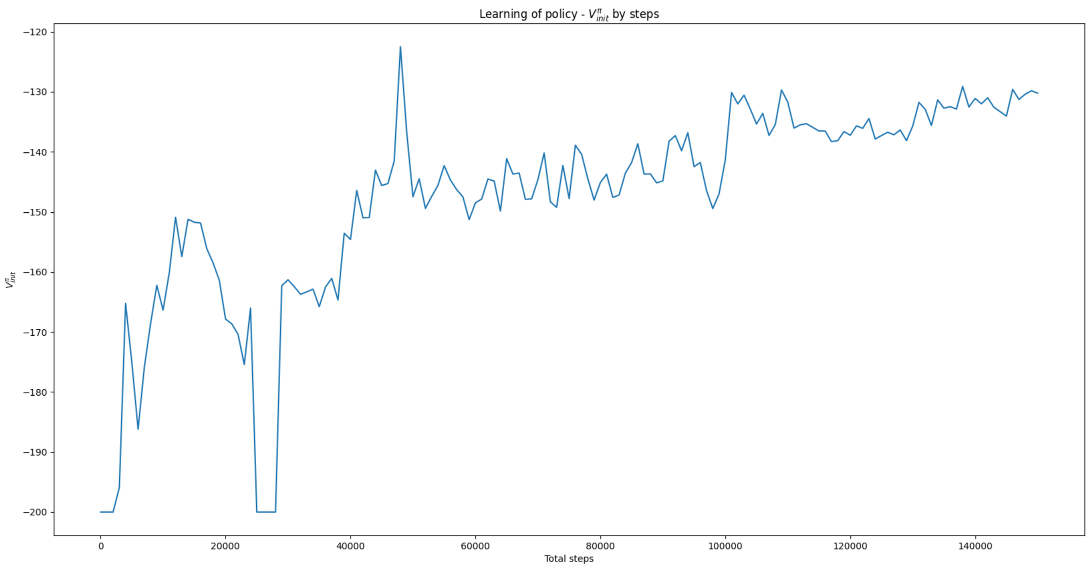

Figure 1: Original problem 200 steps limit per episode