Exercise 1

Eugeniu Vezeteu - 886240
ELEC-E8125 - Reinforcement Learning

September 20, 2020

1 Task 1

In this task the agent was trained with 200 timesteps per episode, for 500 episode, and tested for 500 timesteps per episode.

```
# Exercise 1
# TODO: For CartPole-v0 - maximum episode length
env._max_episode_steps = 200 #used for training
if args.test is not None:
    env._max_episode_steps = 500 #test with 500 steps per
        episode
```

Reward for training for 500 episodes with 200 steps per episode is presented in Figure. 1.

1.1 Question 1.1

YES(in general) - because the agent is trying to balance the pole as long as possible *(while not done)*, and *done* parameter will be True when the pole falls down or the gym environment *max episode steps* will be over. In our case it may be **Not**, because while balancing, the cartpole may move to left/right side of the screen, and for 200 timesteps it will be on the screen, while for other steps it may not (current cartpole x position may be out of the X range, and gym will return *done* = True).

2 Task 2

On this task the agent was trained several times, from scratch, see Figure. 2. Each time I got different average test rewards, (365.35, 77.14, 128.38, 500).

2.1 Question 1.2

Each time the behavior and the performance of the trained model were different, because there the agent sometimes get stuck in the **local optima**.

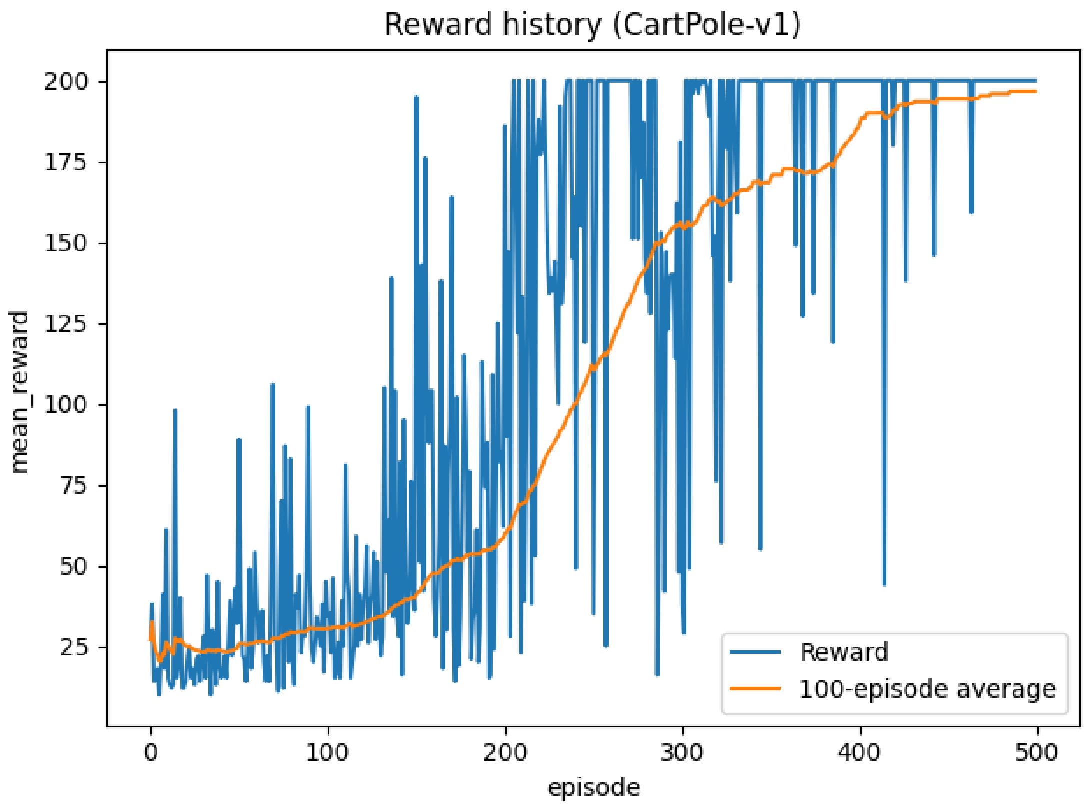


Figure 1: Task 1 train with 200 timesteps for 500 episodes with initial reward function

3 Question 2

There is a high variation is due to local optima, and some agents get stuck there, while other agents get a better performance (find global optima). Stochasticity refers to random actions taken by agent, need a balance between exploration and exploitation.

4 Task 2 - Reward functions

State space is presented in Figure 3.

1. Balance the pole close to the center of the screen (close to x = 0) At this task I computed the distance to 0, and reward the agent, everytime when its close enough to 0, otherwise reward is $\frac{1}{distance-to-0}$.

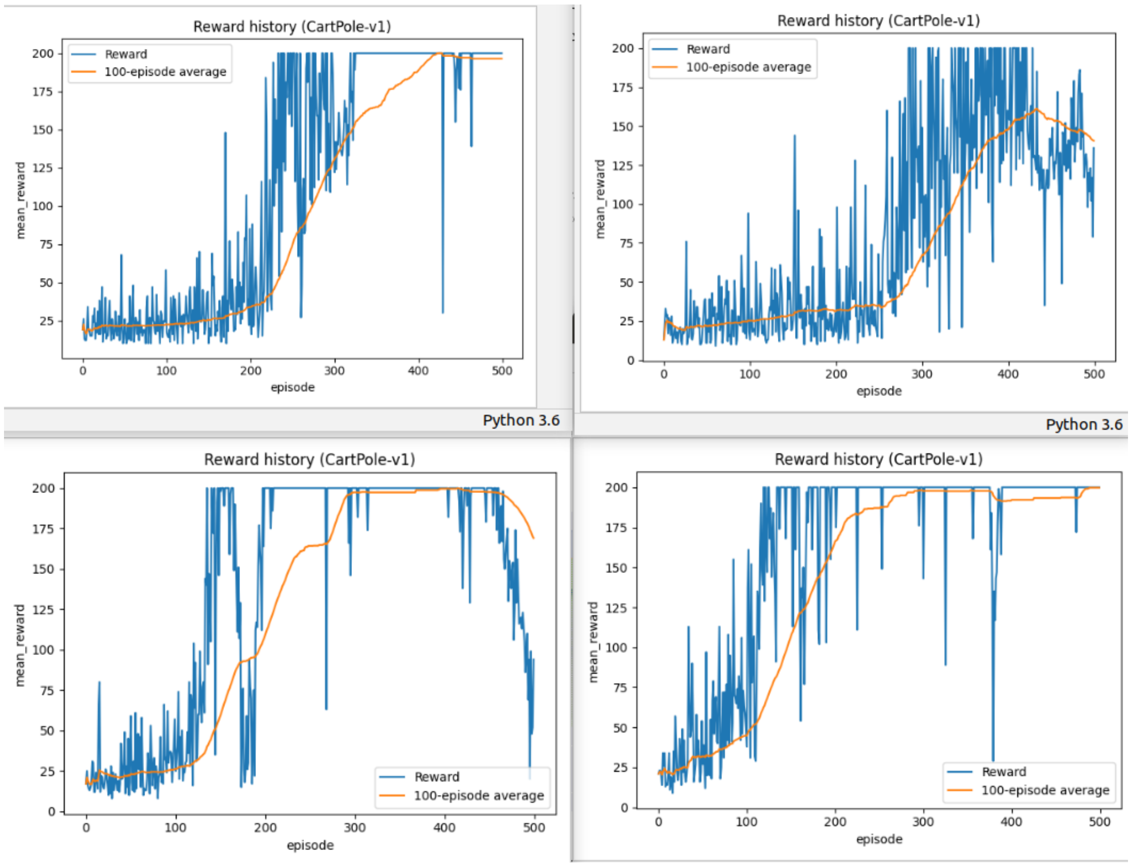
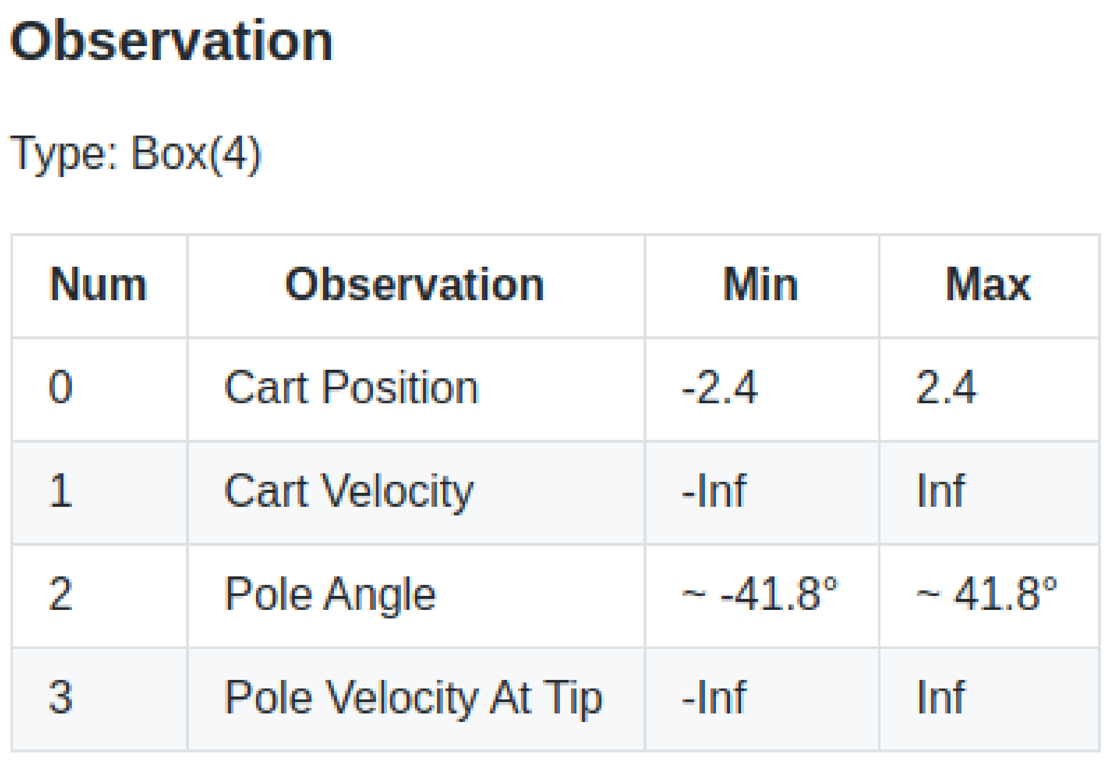


Figure 2: Task 2 train agent several times from scratch with 200 timesteps for 500 episodes with initial reward function

```python
#Task 2.1 - reward function to balance at x = 0
reward = 0.01  # initial reward
diff = np.abs(x - 0)  # compute distance to target
if diff < 0.1:  # if close enough, + reward it again
    reward += 10
else:
    reward += 1/diff
```

The agent was trained for 500 episodes with 300 timesteps for training and 500 timesteps for testing. To test it please use the file named *CartPole-v2_0_params.ai*, it should balance the pole at position x=0
Training result is presented in Figure 4.

2. Balance the pole in an arbitrary point of the screen x = $x_{desired}$
   In this case I computed the distance to the target, normalize it to [0-1], reward the agent if it's closer to the target, the agent also gets reward +2 if its close enough to the target (lets say a threshold 0.01). The agent is penalize then when its out of the screen. Parameter *desired_x* is passed in a command line.



Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Figure 3: CartPole State space, [1].

```python
#Task 2.2 - reward function to balance at x = desired_x
#for full code please see cart_pole.py

reward = 0.001     #initial reward
desired_x = args.desired_x
diff = np.abs(x - desired_x) #compute distance to target
s = (1 - (diff / 4.8)) #normalize diff to be in [0,1] and substract it from 1
reward += s #add this small reward (closer to the target => bigger reward)
if diff < 0.01: #if close enough, + reward it again
    reward += 2
if x < left_most - .2 or x > right_most + .2:  # penalize
    reward -= s
```

Agent was trained for 500 episodes with 400 timesteps for training and 2000 timesteps for testing. To test it please use the file named *CartPole-v2_1_params.ai*, it should balance the pole at position x=-1
Training result is presented in Figure 5.

3. Keep the cart moving from the leftmost to rightmost side of the track as fast as possible
On this task I used the previous idea, of incentivise the agent to achieve some desired

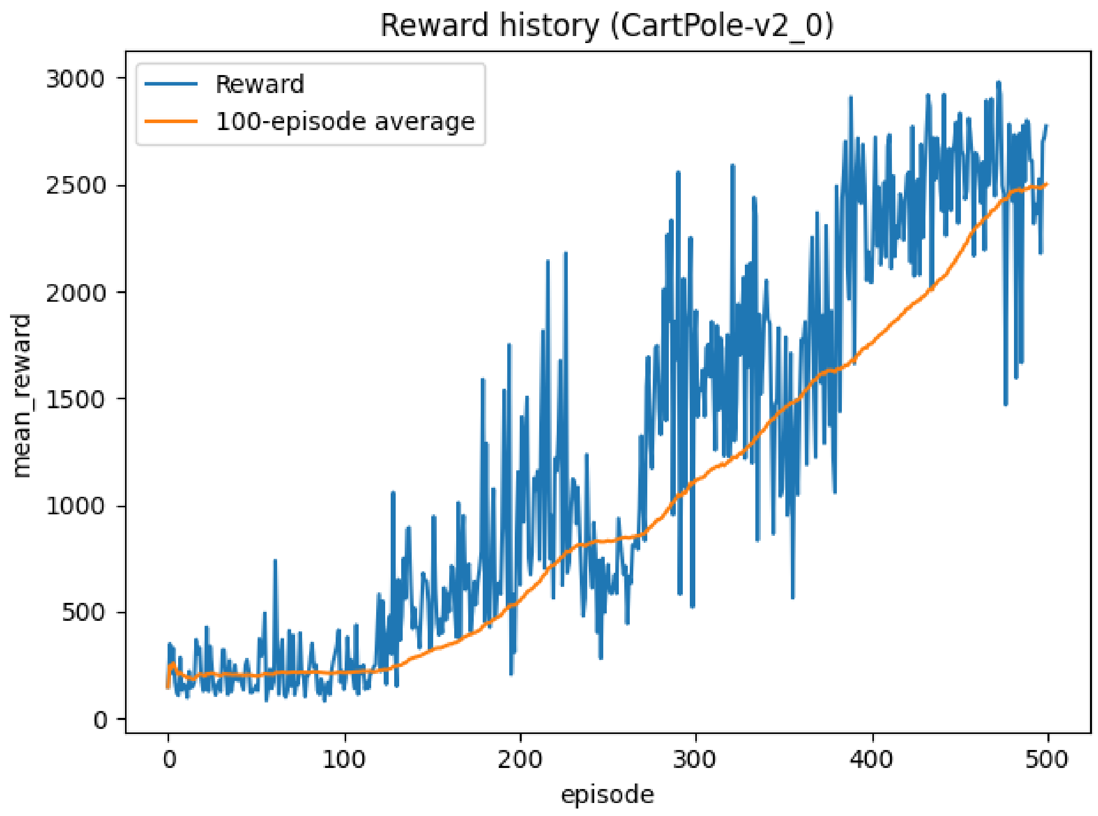


Figure 4: Task 2.1 Balance the pole position x=0 - training

position. $leftmost, rightmost = -1.9, 1.9$ were defined as 2 targets to achieve. The idea is that the agent is achieving reward if he is on the left(right) side, and the target is switched to right(left) side. Also the agent is rewarded if it get closer to the target.

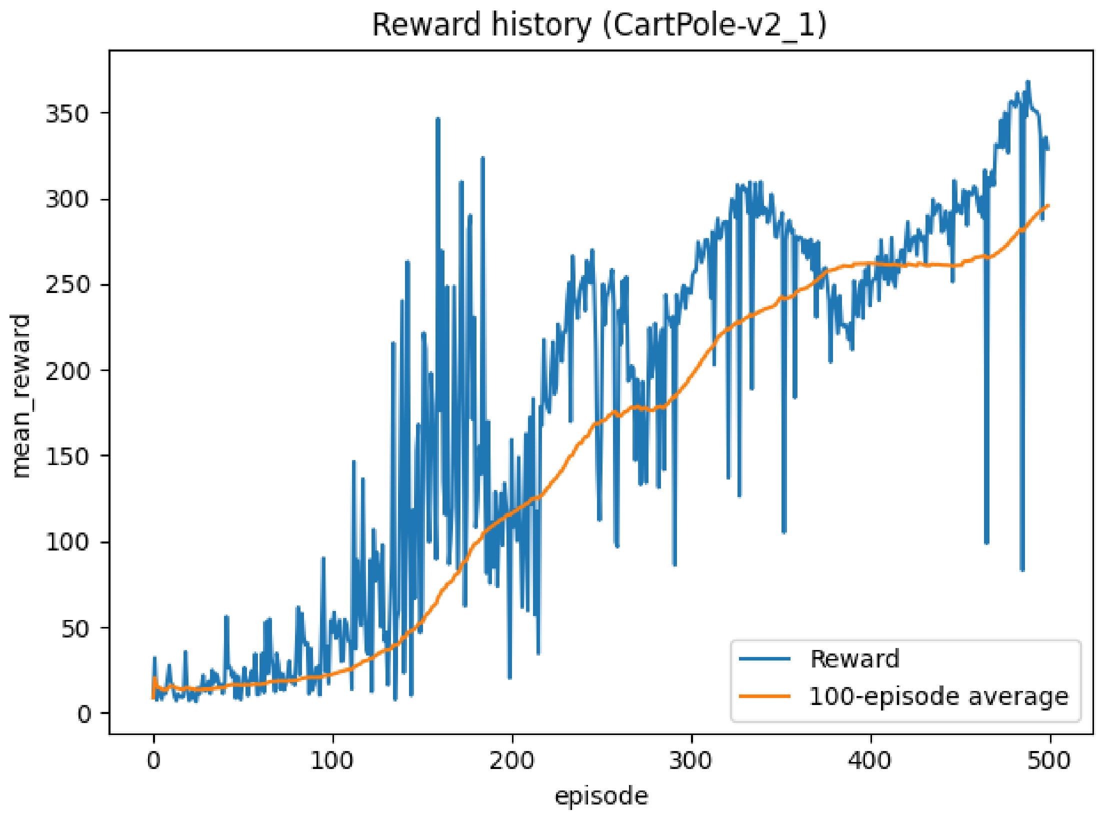


Figure 5: Task 2.2 Balance the pole in an arbitrary point of the screen - training

```
#Task 2.3 - reward function  first part
#moving from the leftmost to rightmost ASAP
#for full code please see cart_pole.py

reward = 0.001 #set some init reward (small number)
desired_x = agent.desired_x  #set my desired x
if x >= right_most: #if the car is on the right, set
   objective to go left
    desired_x = left_most
    reward += 1.
elif x<= left_most:  #if car is on the left, set objective
   to go right
    desired_x = right_most
    reward += 1.
diff = np.abs(x - desired_x)  # compute distance to target
s = (1 - (diff / 4.8))  # normalize diff to be in [0,1]
reward += s  #add reward(closer to target => bigger reward)
if diff < 0.01:  # if close enough
    reward += 1.
# penalize if out of the range
if x <= left_most-.2 or x >= right_most+.2:
    reward -= 5.
```

To move ASAP, the agent is also rewarded based on it's velocity. The limits of the velocity are $(-\infty, \infty)$. So, in order to normalize it, I used *deque* to keep track of the last 100 velocities.

```
#Task 2.3 - reward function - second part
#moving from the leftmost to rightmost ASAP
#for full code please see cart_pole.py

from collections import deque
velocities = deque(maxlen=100)
velocities.append(0)

#all code here , see cart_pole.py
...

curr_speed = np.abs(curr_speed)
velocities.append(curr_speed)   #keep track the car
    velocity
normalized_vel = (curr_speed - min(velocities)) / (max(
    velocities) - min(velocities)) #normalize vel
reward += normalized_vel
```

Agent is rewarded based on distance to target, and on the velocity, and it is penalized if is out of the screen.
Resut on training with this reward function is presented in Figure 6.

I got this fluctuations in average reward, because the agent learns to achieve one side of the screen, and then the target is switched. Training was performed for 1000 episodes, each episode with 500 timesteps, for testing the timesteps were increased to 2000 to see the behaviour. To test it please use the file named *CartPole-v2_2_params.ai*

5 Task 3

The agent with the third reward function was trained 15 times, using *multiple_cartpoles.py*, training results are presented in Figure 7.

6 Question 3

The agent starts in the middle of the screen, firstly it goes to the left most and then to right most, and keep balancing for the number of given timesteps. Sometimes the agent goes too far left or right and can go beyond X limits, (this rarely happens). The highest velocity that the agent reaches is 1.33, see Figure 8.

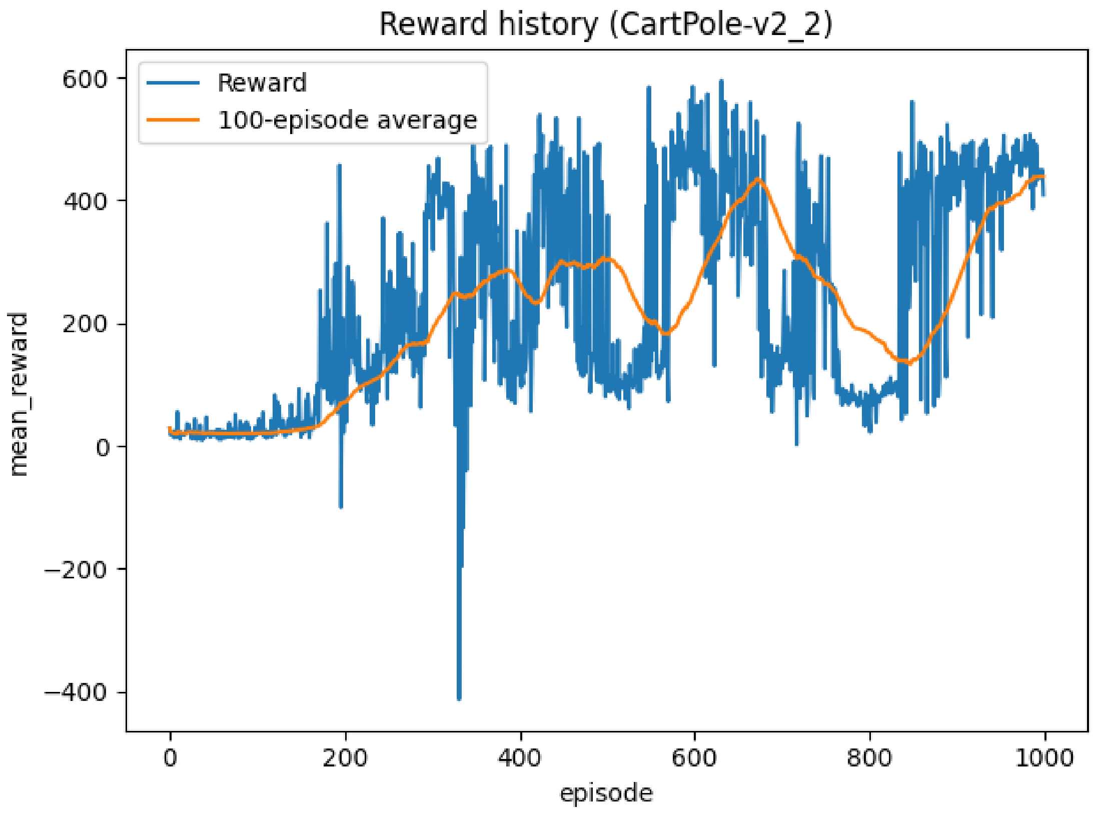


Figure 6: Task 2.3 Cart moving from the leftmost to rightmost side of the track as fast as possible - training

References

[1] "Cartpole v0." `https://github.com/openai/gym/wiki/CartPole-v0`. Accessed: 2020-09-12.

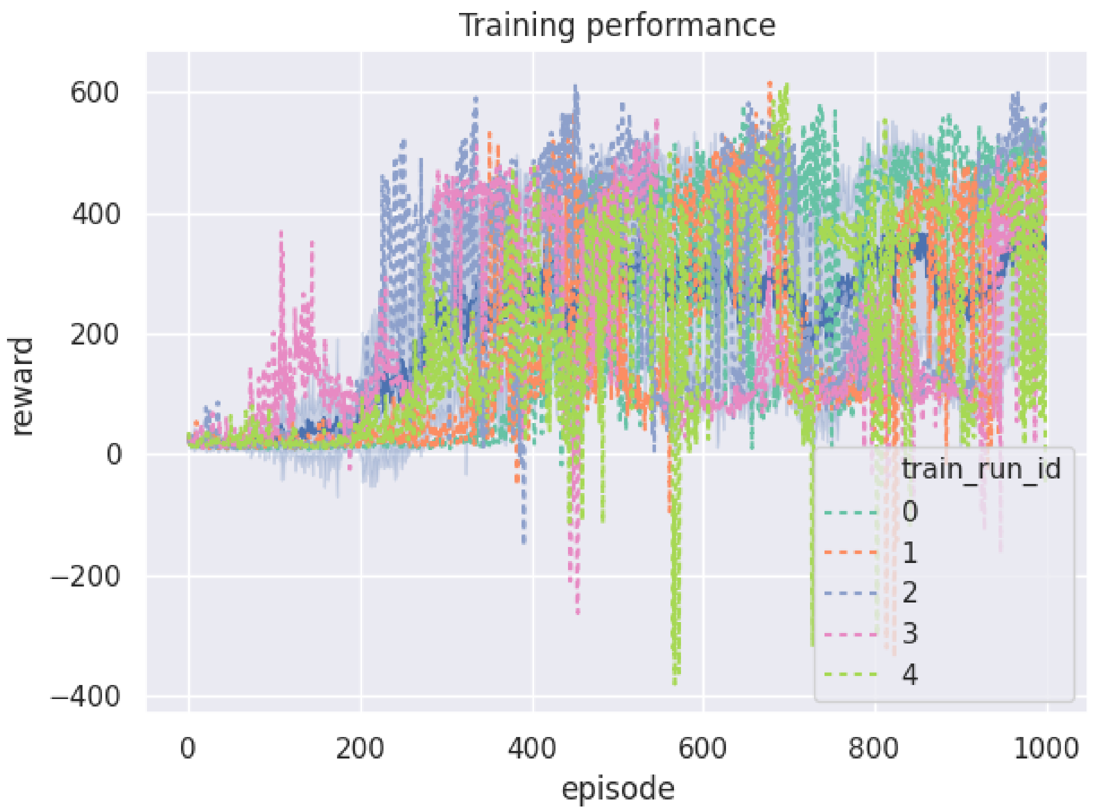


Figure 7: Task 3. Training agent 15 times with third reward function

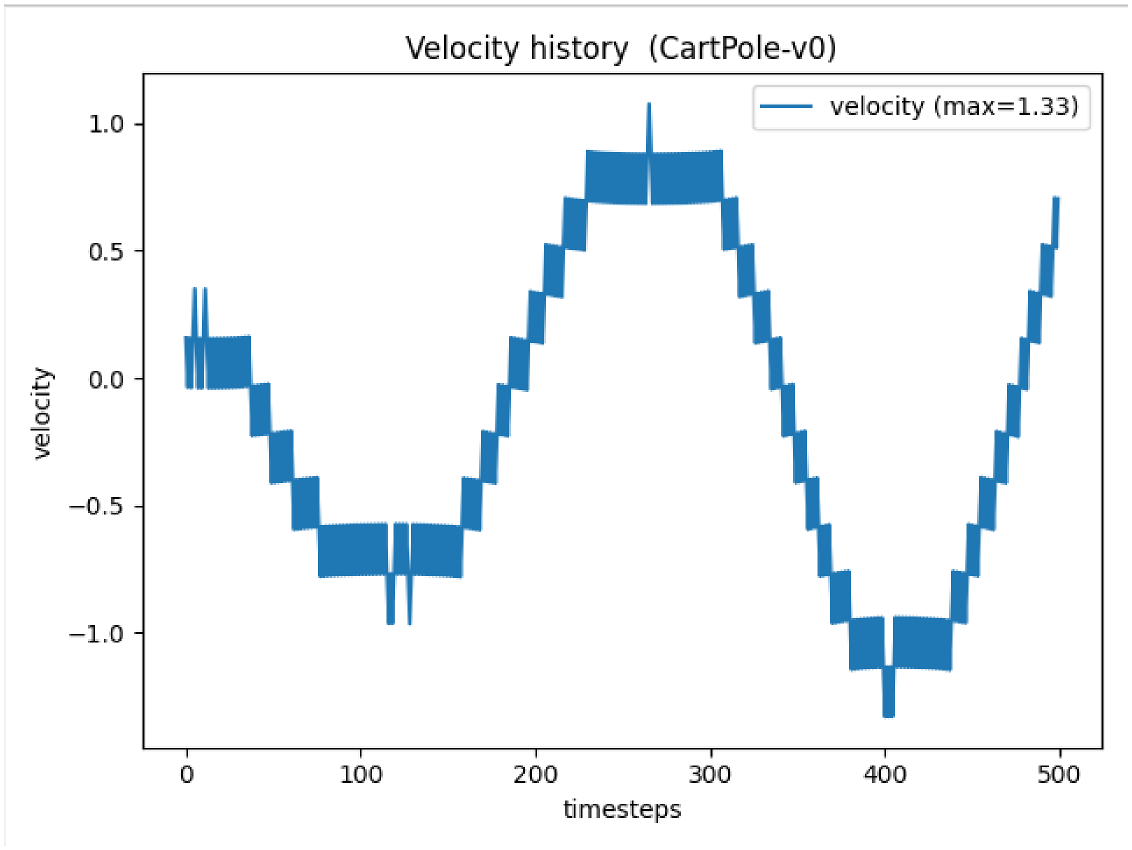


Figure 8: Task 3.3. Testing agent for 500 timesteps with the third reward function