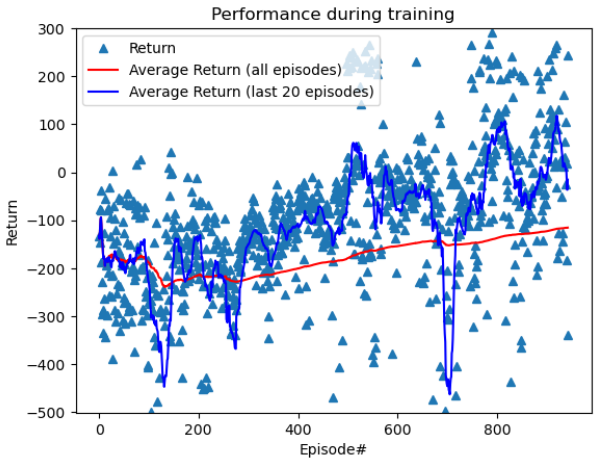
Assignment 4

Kevin McCoy

1. Deep Deterministic Policy Gradient

1.1 30pts

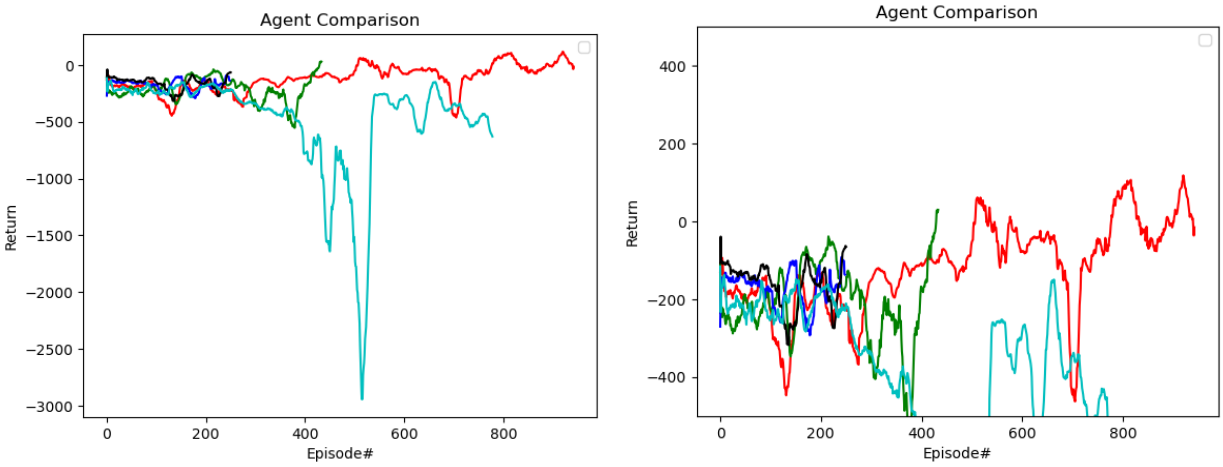
The DDPG implementation slowly learns the optimal policy, albeit very slowly. The algorithm is also prone to sudden dips / valleys in performance, leading to a slow overall learning. I expect that given enough time, and a decay of the exploration noise, the model would perform better.



1.2 30pts

The performances of five different models, with the same initial set up, are shown below. Note that two plots show the plots at different scales. This was necessary as one agent temporarily diverged and experienced very low rewards. I'm not quite sure how this happened, but I hypothesize that the architecture of the neural networks has something to do with it.

The variance can be caused by a few different things. First, the model learning rates may get too small before optimal learning, possibly leading to settling into a locally optimal policy. The state and action spaces are also complex enough, that the model may not have time to fully explore them before this happens. I believe an optimization of my network parameters and architecture could help. Anecdotally, my model was performing very poorly until I implemented a ReLU function after my linear layers. It also helped to apply a tanh layer to the policy, ensuring that all actions were between -1 and 1. I think the concepts of Behavioral cloning could also be very effective here. Even just one run of an expert landing the lunar lander could prove beneficial. I am afraid that the model never has a chance to land the lunar craft at all, and possibly just avoid crashing, before the model converges.



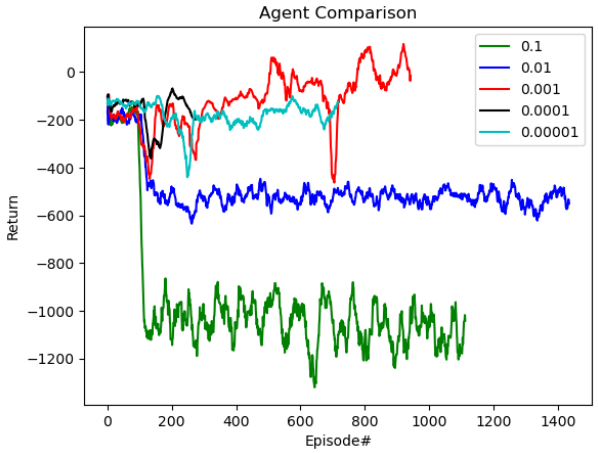
(a) Zoomed Out Image

(b) Zoomed In Image

Figure 1: Performance of 5 Models

1.3 15pts

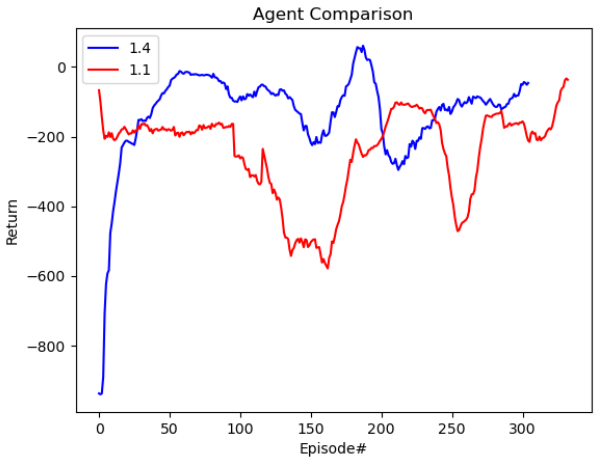
I decided to try changing the learning rates of both the critic and actor networks. The initial learning rate of 0.001. The method did especially poorly with a large learning rate, which likely caused the neural network to diverge. The dropoff at 100 steps is where the warmup period ends. This is noteworthy as the model does worse after beginning training then before with the initially random weights!



1.4 10pts

The off-the-shelf method from Stable Baselines 3 starts off worse, but quickly improves to equal or better performance. This make sense as my neural network probably got lucky with its initial weights. It is also possible that the SB3 explores more heavily than my algorithm.

[5in]


Agent Comparison
1.4
1.1
0
−200
−400
−600
−800
Return
0
50
100
150
200
250
300
Episode#

2. Principle of Maximum Entropy

2.1 5pts

Provide the mathematical definition of entropy H(p) of a distribution p(x).

$$H(p) = \mathbb{E}[-log\ p(x)]$$

2.2 10pts

Let us denote "Pr(turn left)" as p_l, "Pr(turn right)" as p_r, and "Pr(go straight)" as p_s.

We are told that $p_l + p_r = p_s$, and we know that $p_l + p_r + p_s = 1$ must be true. Thus it is obvious that $p_s = \frac{1}{2}$, but we don't know what the values of p_l or p_r are, only that $p_l + p_r = 0.5$. We can write the entropy as:

$$H(p) = -\sum_i p_i * \log(p_i) \tag{1}$$

$$= -\left[p_l(\log(p_l)) + p_r(\log(p_r)) + \frac{1}{2}(\log(\frac{1}{2}))\right] \tag{2}$$

To find the maximizer of entropy, we can solve the Lagrangian

$$L = H(p) + \lambda(p_l + p_r - 0.5) = -\left[p_l(\log(p_l)) + p_r(\log(p_r)) + \frac{1}{2}(\log(\frac{1}{2}))\right] + \lambda(p_l + p_r - 0.5)$$

We can know solve by calculating:

$$\frac{\partial L}{\partial p_l} = -\log(p_l) - 1 + \lambda$$

$$\frac{\partial L}{\partial p_r} = -\log(p_r) - 1 + \lambda$$

Setting these both to zero yields the intuitive answer that $p_l = p_r = 0.25$.