ECEN 689 Reinforcement Learning Assignment-4 Report

All variants of the policy gradient are reviewed thoroughly and implemented in the following questions.

Got familiar to RL coding pipeline.

Variants of policy gradient that should be implemented

$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau \sim P_{\text{traj},\theta}(\cdot)}\left[R(\tau)\sum_{t=0}^{\infty}\nabla \log \pi_\theta\left(x_t, a_t\right)\right], \tag{1}$$

$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau \sim P_{\text{traj},}\ (\tau)}\left[\sum_{t=0}^{\infty}\gamma^t G_t \nabla \log \pi_\theta\left(x_t, a_t\right)\right], \tag{2}$$

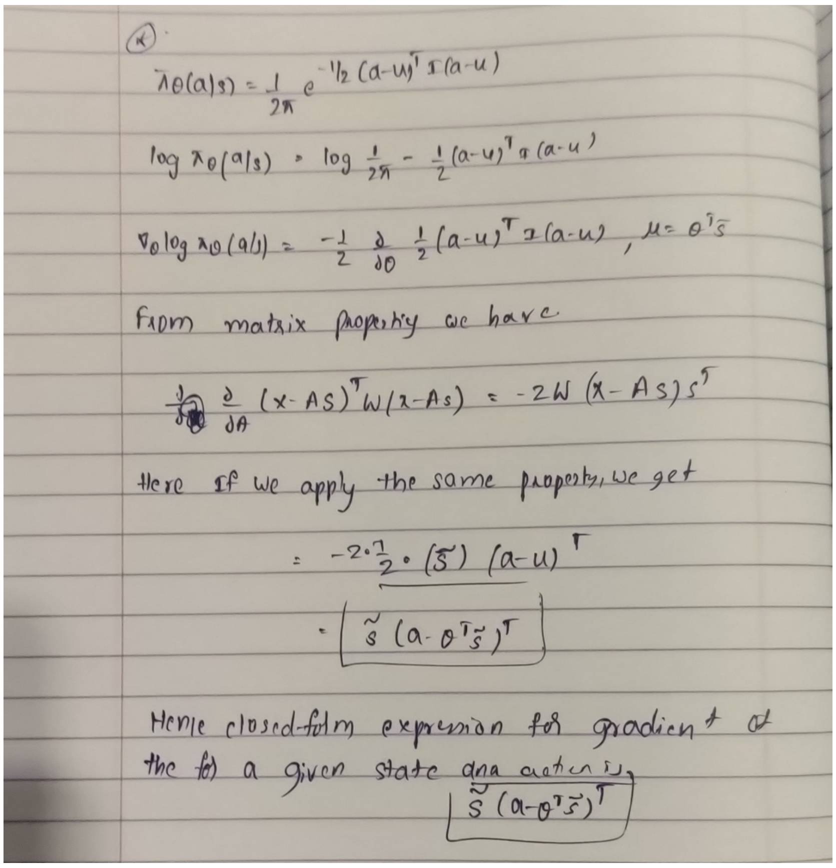
$$\nabla V_{\pi_\theta} = \mathbb{E}_{\tau \sim P_{\text{traj},}\ (\tau)}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi_\theta}(x_t, a_t)\nabla \log \pi_\theta(x_t, a_t)\right]. \tag{3}$$

Question 3.1.1 : Recall the PDF ofπθ given in the environment description. Analytically compute a closed form expression for the gradient of the policy for a given action and state ,i.e., gradient of the term ∇θlogπθ(a|s).Include the expression in your submission report.

Solution : Closed form expression for the gradient of the policy for a given action and state is

$$(\tilde{s})(a - \theta^T\tilde{s})^T$$

The formulation has been attached in the image below.



(x).

$$\pi_\theta(a|s) = \frac{1}{2\pi} e^{-\frac{1}{2}(a-u)^T I (a-u)}$$

$$\log \pi_\theta(a|s) = \log \frac{1}{2\pi} - \frac{1}{2}(a-u)^T I (a-u)$$

$$\nabla_\theta \log \pi_\theta(a|s) = -\frac{1}{2} \frac{\partial}{\partial \theta} \frac{1}{2}(a-u)^T I (a-u) \quad , \mu = \theta^T \tilde{s}$$

From matrix property we have

$$\frac{\partial}{\partial A} (x - As)^T W (x - As) = -2W(x - As)s^T$$

Here If we apply the same property, we get

$$= -2 \cdot \frac{1}{2} \cdot (\tilde{s}) (a-u)^T$$

$$= \boxed{\tilde{s}(a - \theta^T \tilde{s})^T}$$

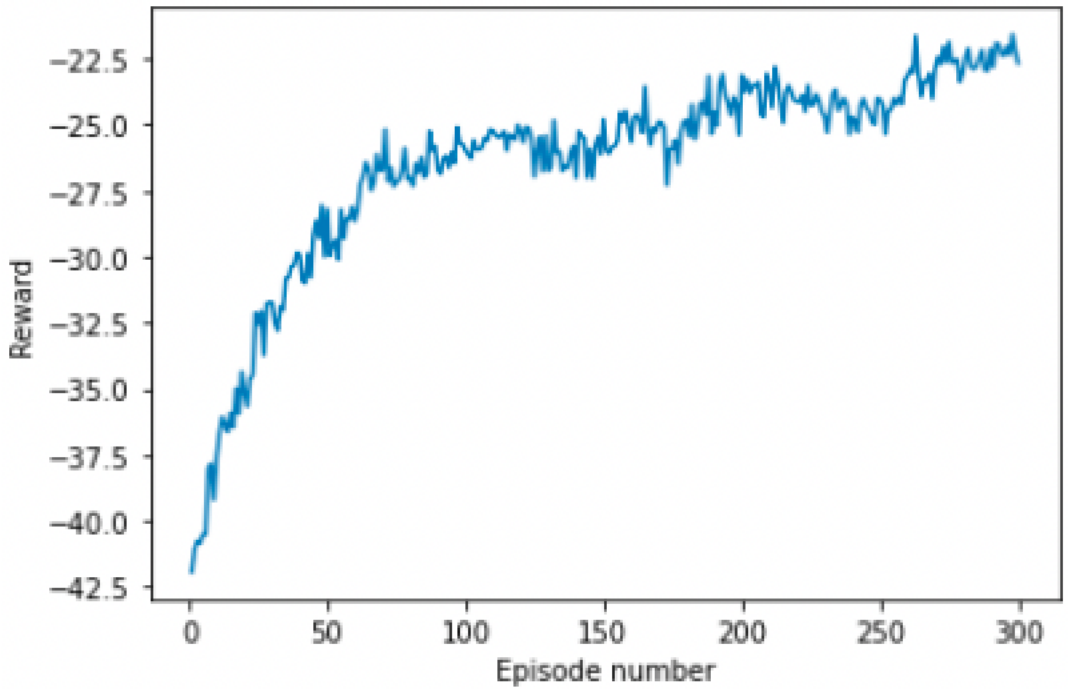
Hence closed-form expression for gradient of the ~~for~~ a given state and action is,

$$\boxed{\tilde{s}(a - \theta^T \tilde{s})^T}$$

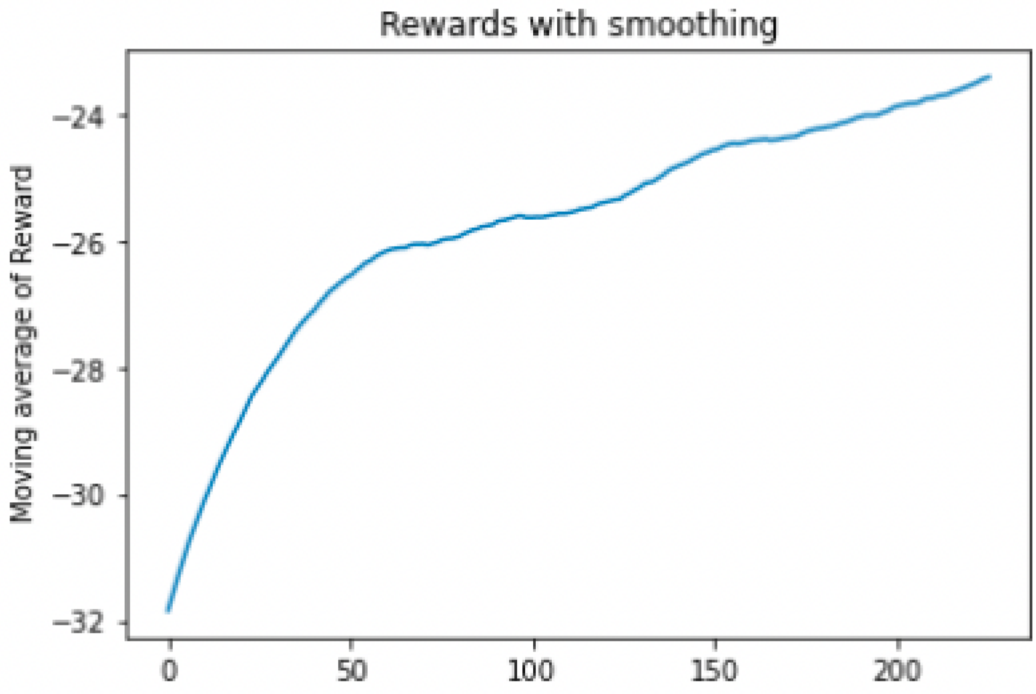
Question 3.1.2 : Implement policy gradient variants specified in Eqn.1and Eqn.2for Point-v0. For this part you would have to update the following files:
(a)mainloop.updateparamsmethod inmain.py.
(b)estimatenetgradmethodsolutions/pointmasssolutions.py
Please feel free to modify the method arguments as needed. We have only provided a basic outline. We recommend a learning rate of~0.1 for this environment. Plot the training curves for each of these variants (discounted sum of rewards per iteration of the training procedure). Summarize your observations. How fast does the algorithm learn? What average return do we see after 100 iterations of the algorithm? Include a rendering of the final policy

Solution : The following files a, b are updated and implemented.

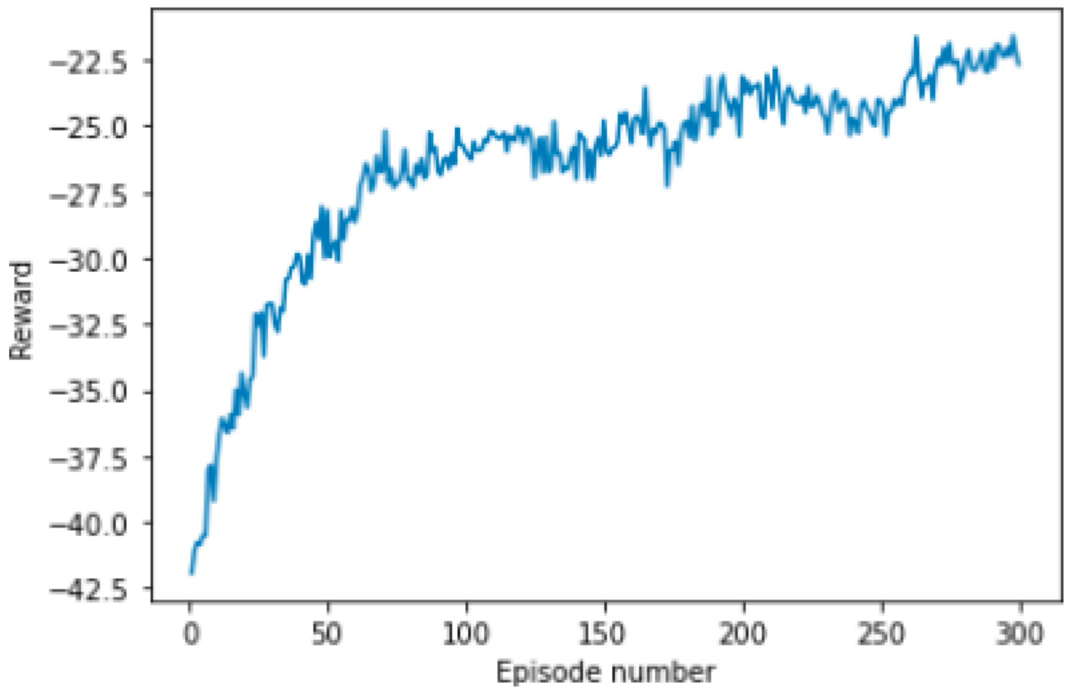
Plot of discounted sum of rewards per iteration for policy gradient with rewards (equation 1):



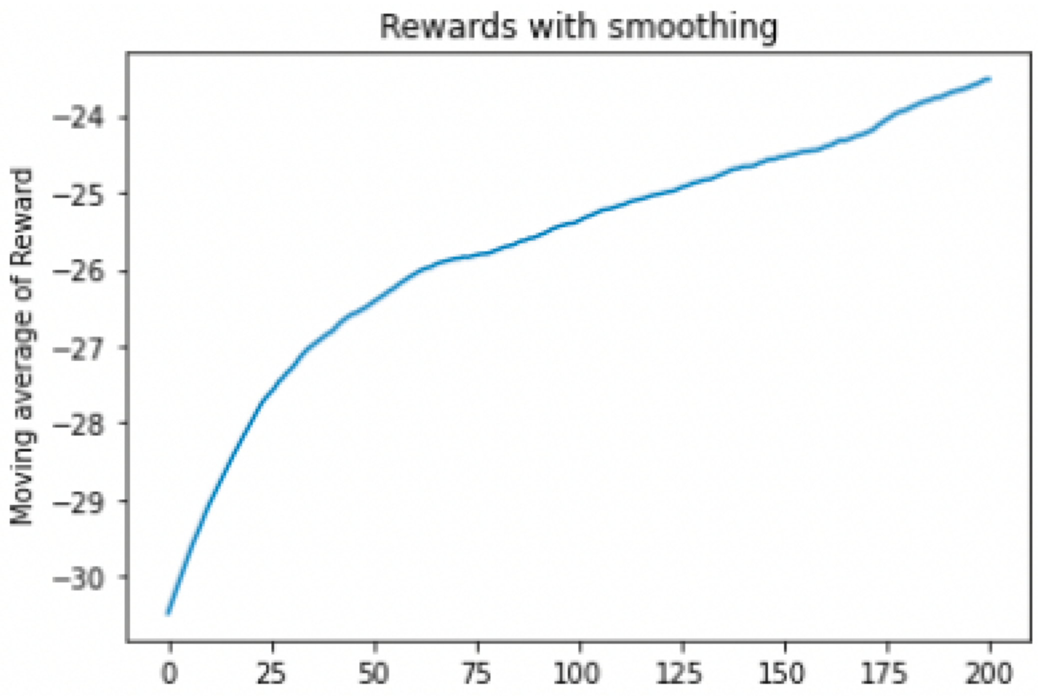
Plot of discounted sum of rewards per iteration for policy gradient with smooth rewards (equation 1):



Plot of discounted sum of rewards per iteration for policy gradient with to-go returns (equation 2):



Plot of discounted sum of rewards per iteration for policy gradient with smooth to go returns (equation 2):



Observations:
It can be observed from the plots that the algorithm implemented with equation 2 took lesser episodes to converge to a better value. Hence, we can say that policy gradient with to-go returns performs better than the equation with rewards. The learning is also smooth and has less variance in case of the algorithm with to-go returns.

Time taken by the algorithm:
For the algorithm implemented with equation 1 using rewards, algorithm converged at a reward value of -24.5. It took 240 episodes to converge.

Reward value is around -22.9 after 400 episodes.

For the algorithm implemented with equation 1 using returns, algorithm converged at a reward value of -23.0. It took 150 episodes to converge.

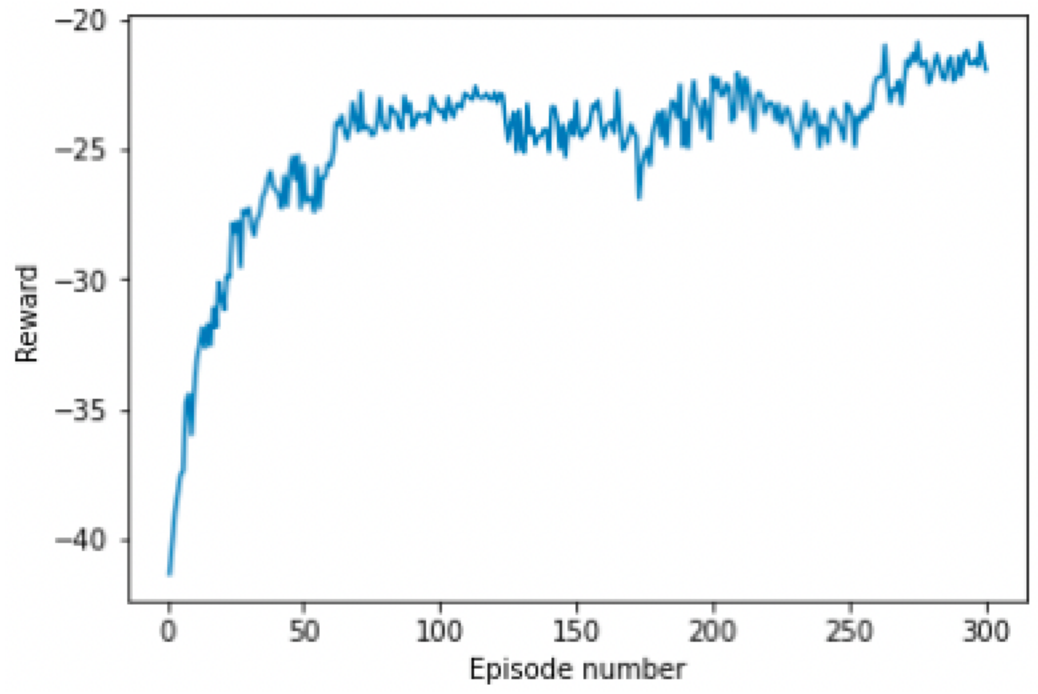
Reward value is around -20.2 after 300 episodes.

Hence, Average return we get after 100 iterations of the algorithm for equation 1 using rewards is : -24.5 and average return we get after 100 iterations of the algorithm for equation 1 using rewards is : -23.0

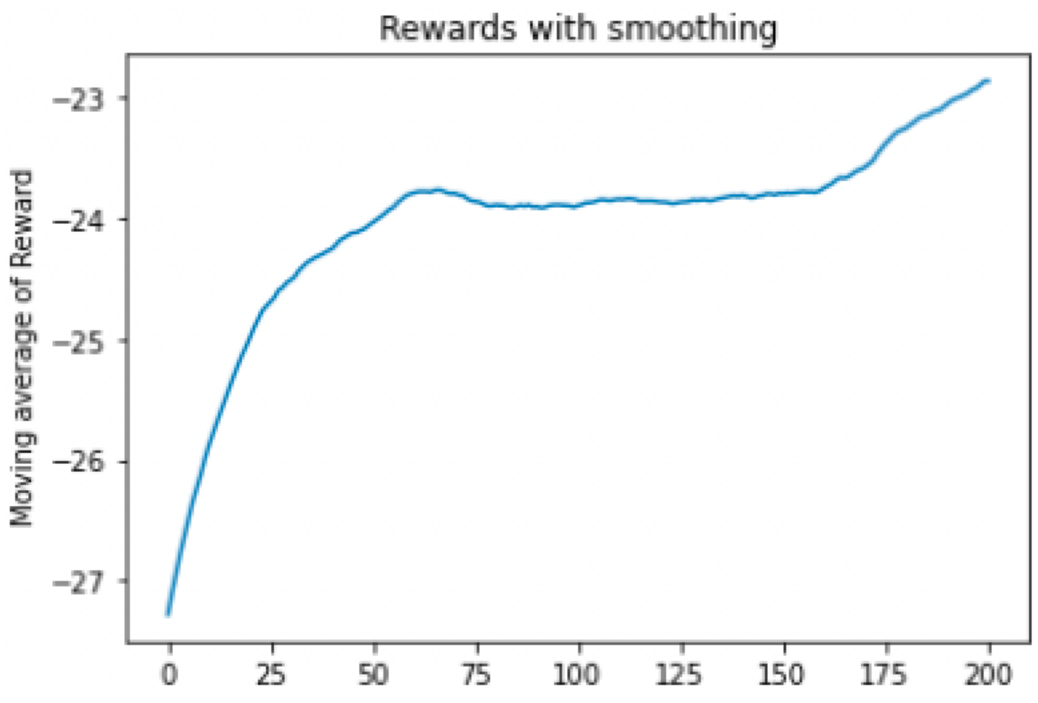
Rendering of the final policy with videos for both equations is attached in the zipped file.

Bonus question: Do you think including a baseline would improve the performance of these algorithms. Try implementing a simple time dependent baseline as follows: Simply set the baseline to the average of all the returns ($R(\tau)$) among the trajectories collected from the previous iteration. If there are no trajectories long enough at a certain time step, we can set the baseline to 0.
Solution:
Yes, I feel including a baseline will improve the performance of the algorithms.
Baseline is set to the average of all returns for all the trajectories collected from the past iteration, else there are less trajectories then baseline is set to 0

Plot of discounted sum of rewards per iteration for policy gradient with rewards (equation 1) using baseline :



Plot of discounted sum of rewards per iteration for policy gradient with to go returns (equation 2) using baseline :



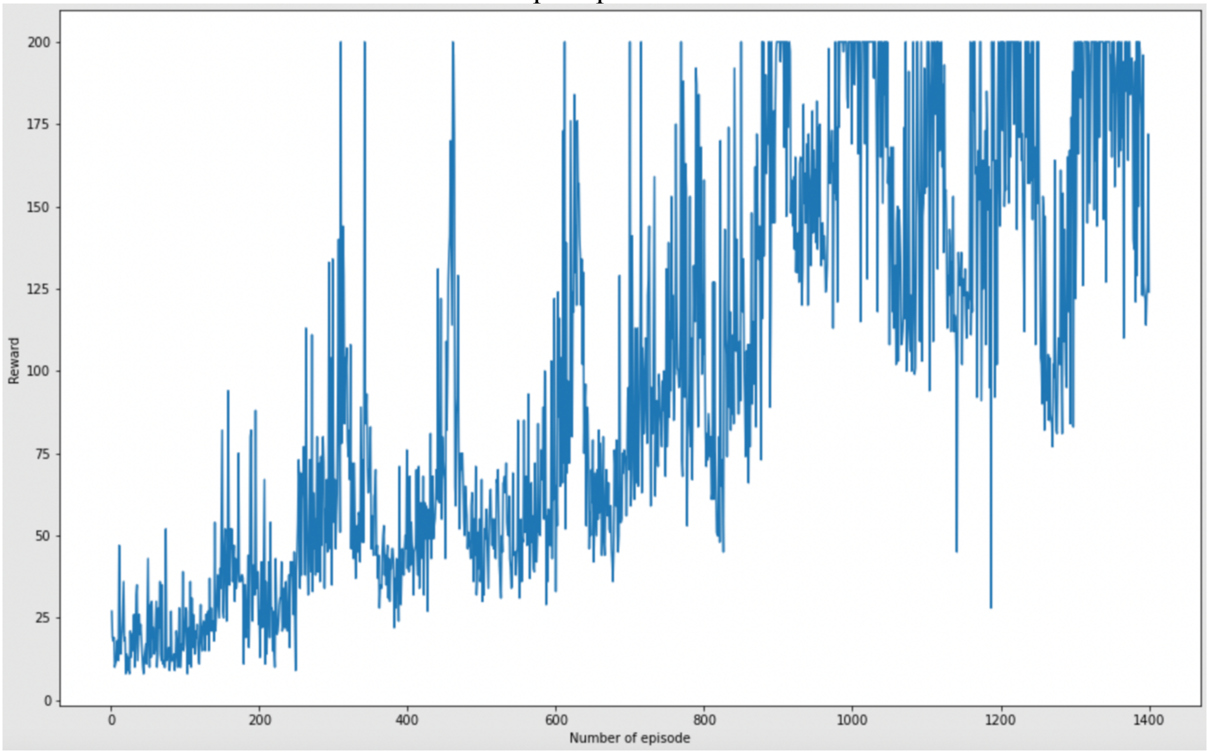
Question 3.2 CartPole-v0:

Question 3.2.1: Implement the REINFORCE Algorithm using the policy gradient described in Equation1on theCartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We do not expect you to solve CartPole-v0 using this approach)

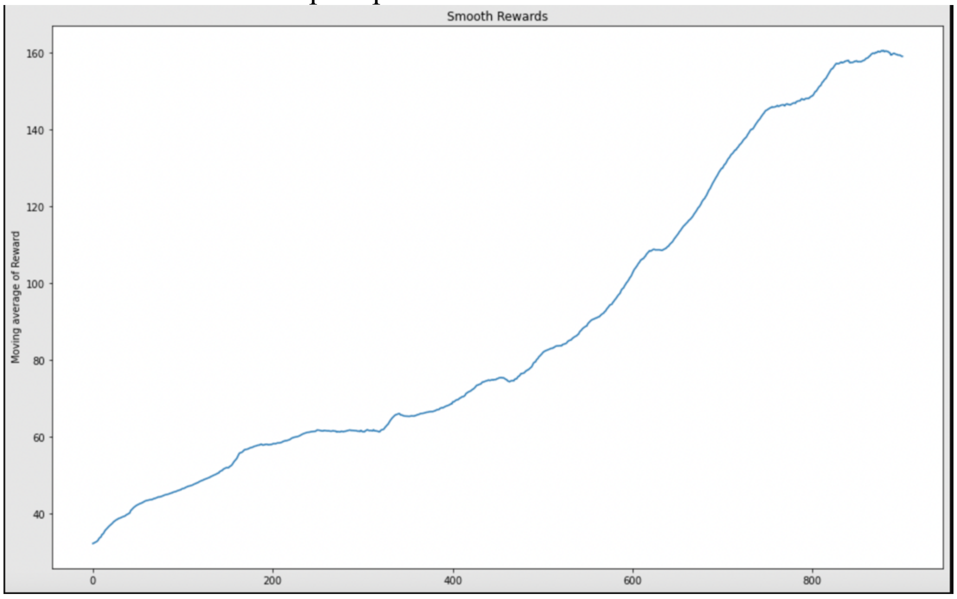
Solution:

The environment Cartpole-v0 is solved in 1437 episodes

Plot of un-discounted cumulative reward per episode is :



Plot of smooth rewards per episode :



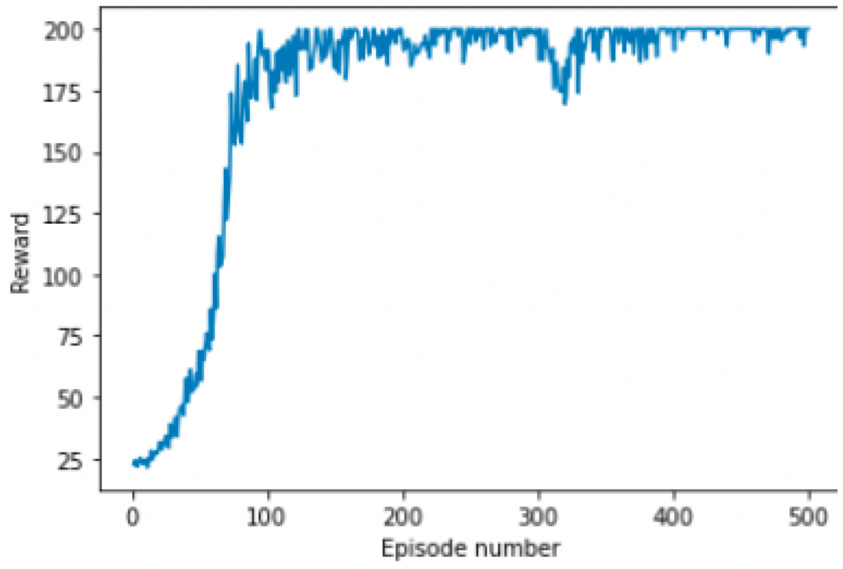
Question 3.2.2 :

Implement the REINFORCE Algorithm using the policy gradient described in Equation2 on theCartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We expect you to solve CartPole-v0 using this approach)

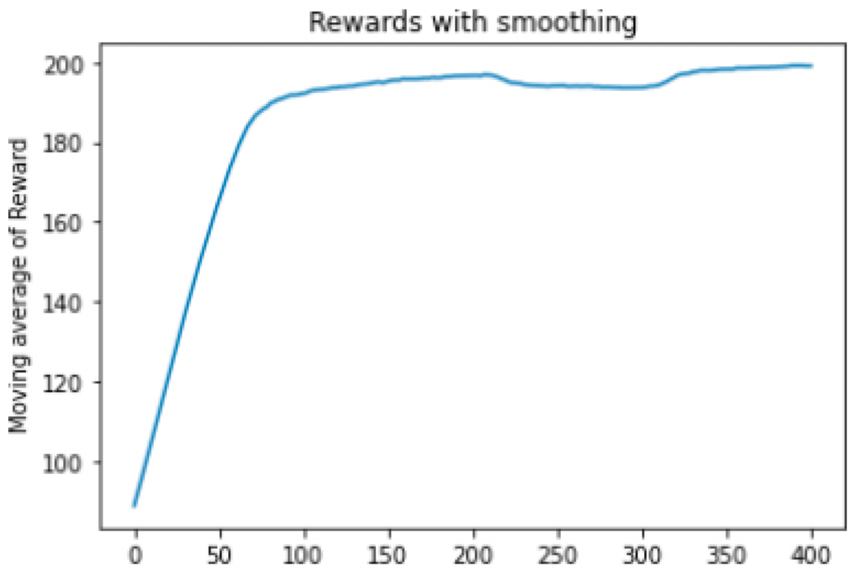
Solution:

Reinforce algorithm is implemented with equation 2 using to-go returns.
With this approach, the environment is solved in 140 episodes. Reward was reaching a constant of 195.0 for 100 episodes.

Plot of un-discounted cumulative reward per episode is :



Plot of smooth rewards per episode:



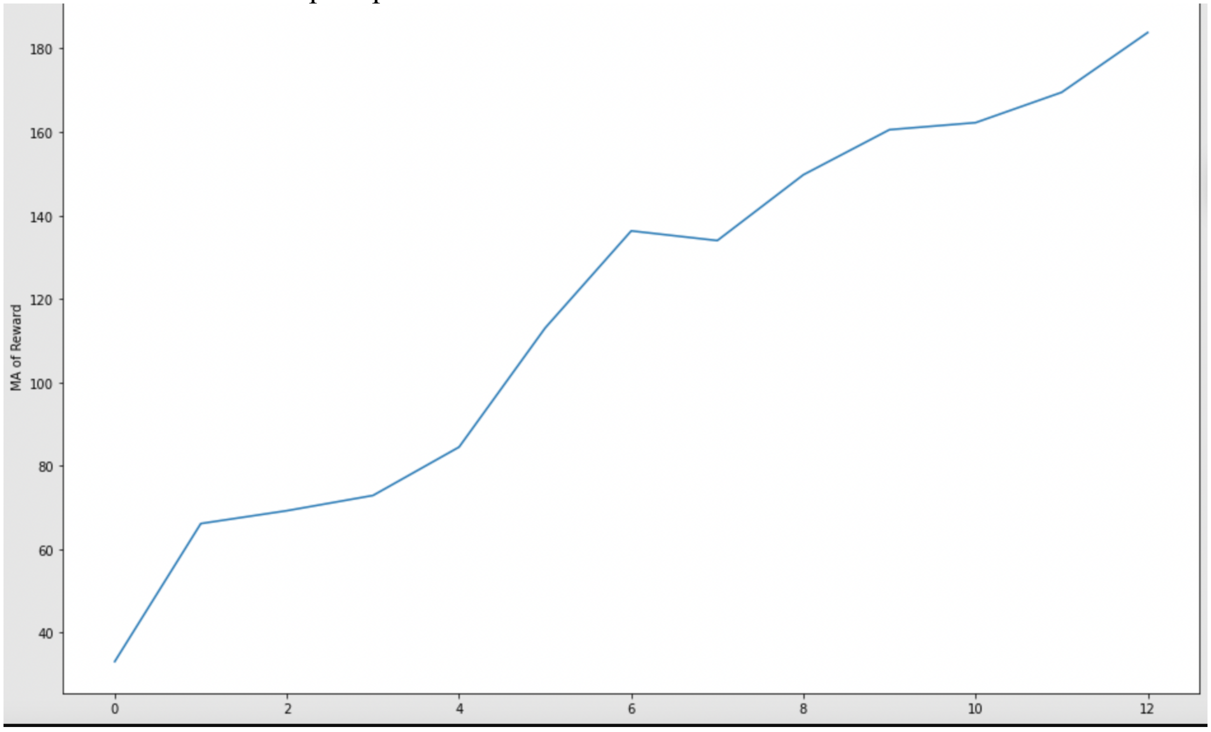
Question 3.2.3:

Implement the REINFORCE Algorithm using the policy gradient described in Equation3on theCartPole-v0 environment. Plot the un-discounted cumulative reward per episode. (Note: We expect you to solve CartPole-v0 using this approach)

Solution:

Here reinforce algorithm is implemented using the Advantages. The advantages code for the pipeline does rigorous parallel processing and requires the assistance of CUDA. Since, I was working in MAC, I couldn't access code needing CUDA and it was throwing me errors. So, I implemented the code from scratch and attached it in cartpole q3.ipynb.

Plot of smooth rewards per episode:



The environment is solved in fewer steps than the previous two approaches. The above graph shows the testing performance after sufficient training.

Question 3.2.4: Summarize your observations on the 3 approaches to implement the REINFORCE algorithm:
Solution :

The Cartpole-V0 environment solved in like 1437 episodes using the first equation of policy gradient with rewards. The environment solved in 140 episodes for the implementation using second equation using to-go returns. Using advantages, the environment is solved in fewer episodes in comparison to the previous two approaches

We can see that the environment is solved quickly and efficiently using the third equation of advantages and it was better using the second approach with returns compared to the first approach with rewards. The learning also becomes smooth using advantages in comparison with returns which has better learning compared to rewards.