CS6700 PROGRAMMING ASSIGNMENT 2

CS21S048
CS21S058

PART 1: DQN

Tasks:

For **DQN**, perform the following (for each environment):

- Start with the set of hyperparameters originally given in tutorial 4 and try solving the environment.
- To account for stochasticity, use the average of 10 runs (at least) for each variation.
- With the goal of improving the agent's performance (choose a metric - could be number of episodes to taken to solve the environment). Try changing a few hyperparameters. List down the new set of parameters and provide reasons for the choice made.
- For each variation of the agent, plot reward curves and print the number of steps to reach the goal in each episode. Use this to comment on the performance of the agent.

- Repeat this till you improve your agent 5 times [OR] try 12 different configurations (at least).
- Make inferences and provide conjectures from all your experiments and results.

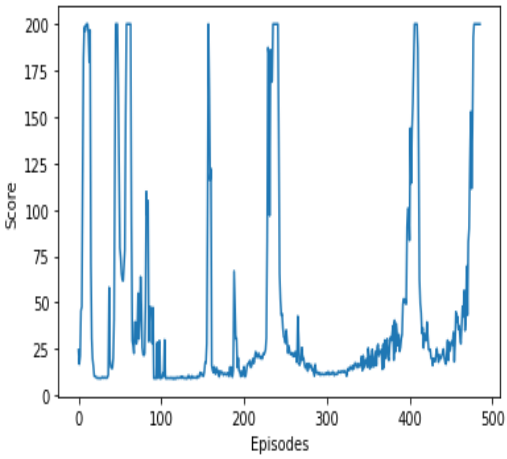
****Clarification on the state of our graphs:**
According to the second point in the tasks required to be done for DQN, it is clearly written to take the average of 10 runs for each variation of hyper-parameters and accordingly we have done that. Taking the average of all the first episodes of the ten experiments, then taking the average of all the second episodes of all the ten experiments and continuing like this. This gives an idea of the average rewards that the same episodes of each experiment is giving. Nowhere in the question

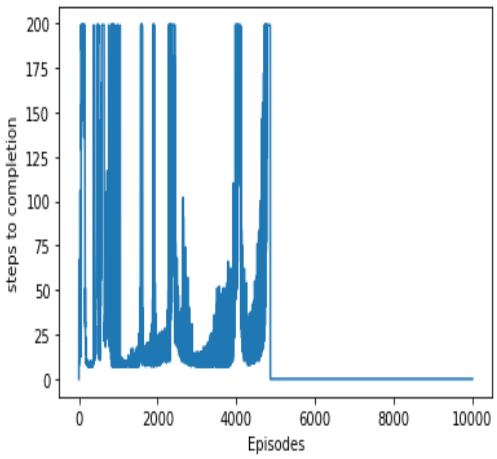
pdf is written we have to take 10 different plots in the same graph and had to expend quite a lot of time and resource to make out what the second task even meant. Our rewards in the graph tend to reach zero at the end of the episodes since the number of episodes required to reach convergence varied and after convergence the reward is taken as zero which on adding with other experiments which has a late convergence pulls down the average rewards. Misunderstanding from both sides led to this condition so I plead you to consider our graphs as the true meaning that they produce gives a different insight into the experiments giving us an idea of how the algorithms work until they converge.

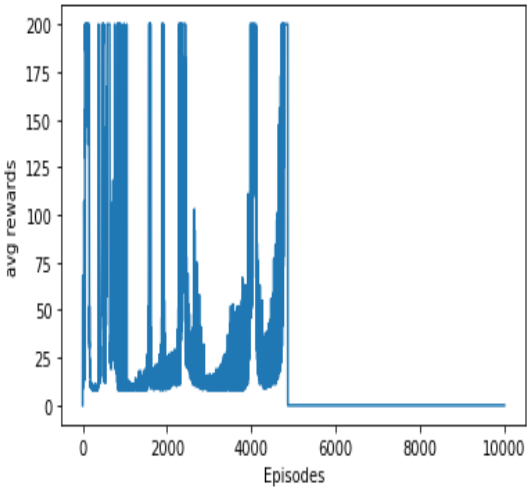
A) CARTPOLE:

1. Tutorial Hyper-parameters:

- Policy: Softmax
- Buffer Size: 10,000
- Batch Size: 64
- Gamma: 0.99
- LR: 0.0005
- Update every: 20
- HL_1: 128
- HL_2: 64
- Beta: 5->0.01(decay=0.995)







Inferences:

Convergences: 10,000, 2243, 1595, 3231, 1437, 5794, 4826, 1374, 1522, 1398

These graphs were the product of the hyper-parameters given in the tutorial for the softmax version. The epsilon greedy version were giving decent results but none of the experiments converged there before 1300 episodes. The lowest there was 1374 episodes and the average episodes required to converge was more than 2400 episodes. For softmax the ten experiments took: 10,000, 2243, 1595, 3231, 1437, 5794, 4826, 1374, 1522, 1398 episodes respectively. It didn't converge once while the epsilon greedy version though giving better convergence for the ones which converged was actually bad at converging. It didn't converge thrice. Our main goal will be to first make all the experiments converge and then try to make them do the same much faster.

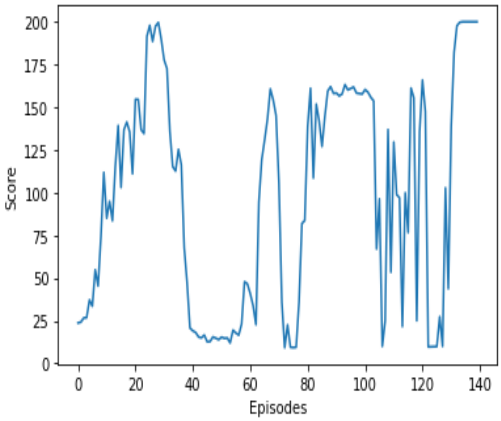
Special Mention: the arrows given beside the hyperparameters will give an idea of how they were different with respect to the last combinations.

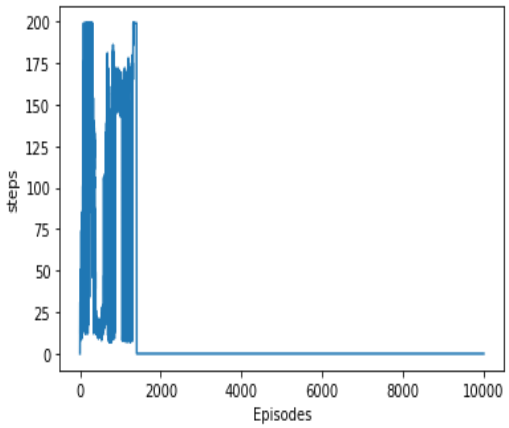
→same as before

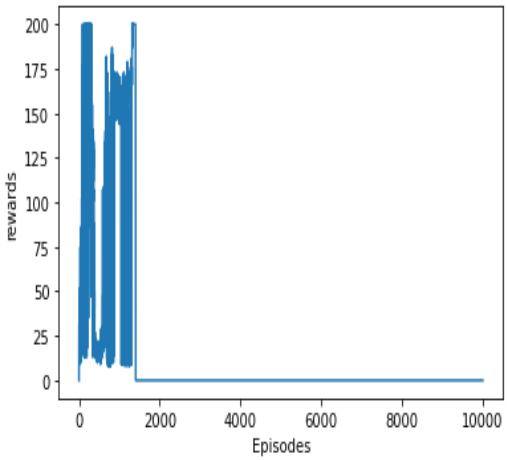
↓ reduced from before
↑ increased from before.
O policy interchange

2.

- Policy: softmax
- Buffer Size: 10,000→
- Batch Size: 64→
- Gamma: 0.99→
- LR=0.00025↓
- Update every: 5↓
- HL_1: 128→
- HL_2: 128↑
- Beta: 8.0->0.01(decay=0.995)↑





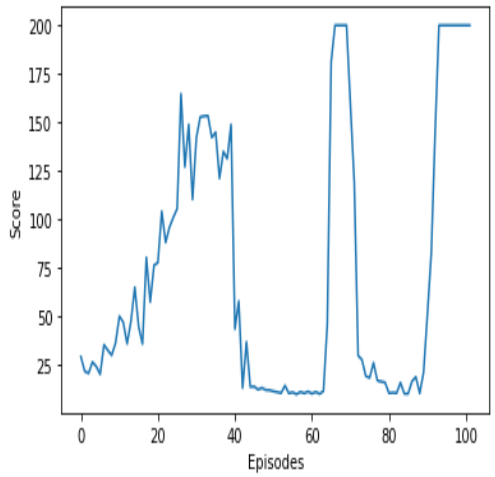


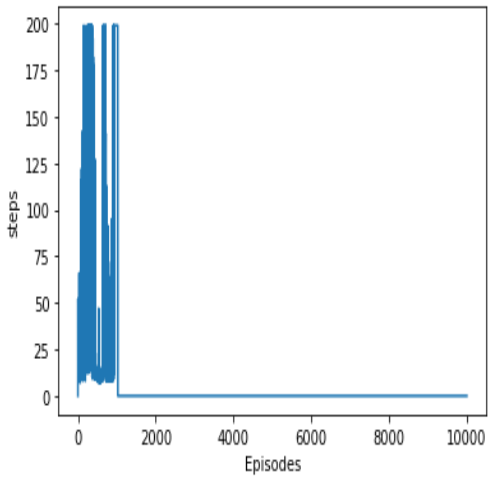
Inferences:

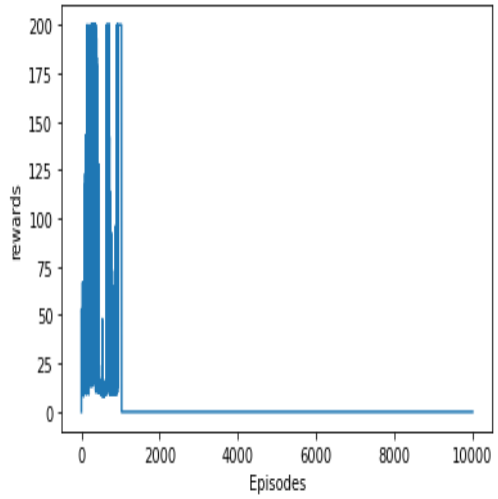
Convergences:- 1734, 2180, 2248, 1668, 10000, 1651, 1119, 1219, 1626, 1064.
One experiment didn't converge. But the episodes required to reach convergence were reduced drastically.
The graph as we can see gets much thinner and much more stable than the graphs made by the softmax version of the tutorial hyper-parameters. That means that the number of episodes required to converge in an average got much lesser. There were actually much more runs between this one and the tutorial one and gave us various insights into how the hyper-parameters were actually influencing the graphs when either working alone or in combination with others. Reducing learning rate for cartpole usually gave worse results when done alone and on a significant level. Changing from 0.0005 to just 0.001 made convergence impossible. Here we are changing only a little from 0.0005 to 0.00025 and getting better results when other hyper-parameter changes were also taken into account. We also changed the second hidden layer and the update every hyper-parameter but it looked like increasing the temperature value for the softmax worked wonders(5->8). After these series of experiments we kept the temperature value(beta) same at 8 as it was the most important change that was made this evolution.

3.

- Policy: epsilon-greedy ↻
- Buffer Size: 10,000→
- Batch Size: 64→
- Gamma: 0.99→
- LR=0.00025→
- Update every: 5→
- HL_1: 128→
- HL_2: 128→
- Epsilon: 1.0->0.01(decay=0.9975)







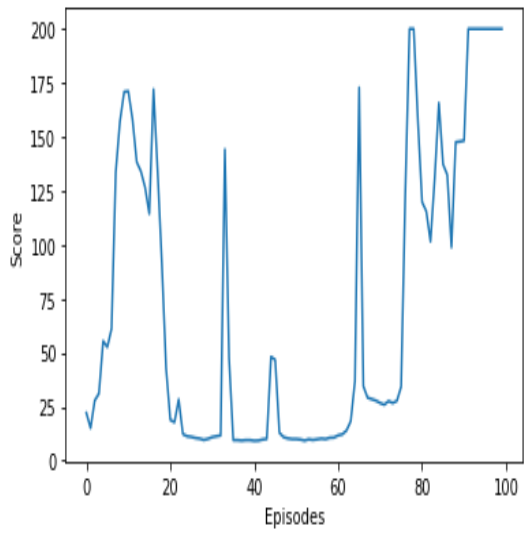
Inferences:

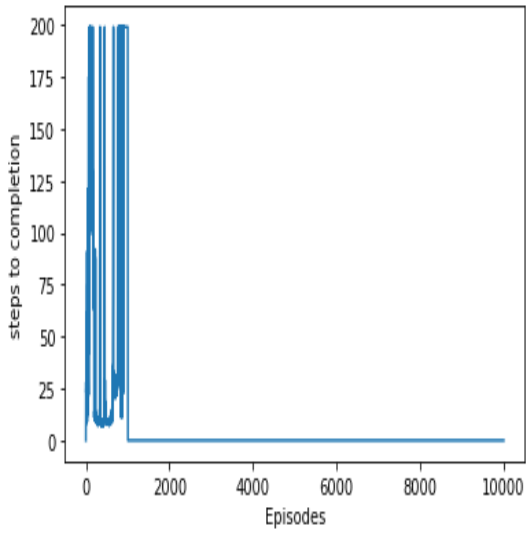
Convergence: 1390,2598,1351,1369,1321,1289,1539,1342,1318,924
Just changing policy with the earlier hyper-parameters gave hugely better rewards, with minimum convergence sitting at 924. Best thing was all the experiments successfully converged. Almost all experiments converged within 1300 episodes with two strays at 2500 and 1500. For all these experiments of Cartpole, epsilon greedy played better role as setting a huge importance on exploitation was important for this environment.

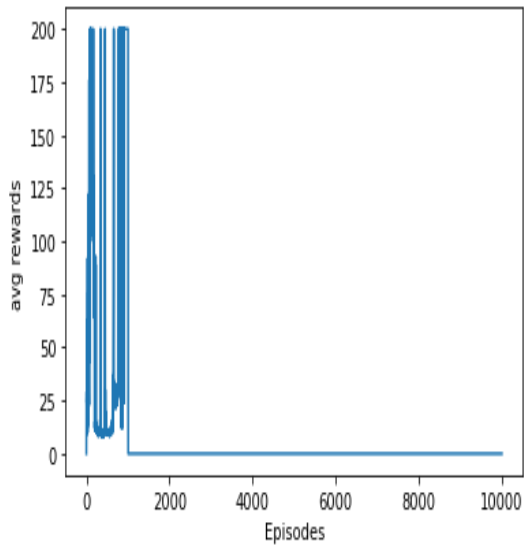
4.

- Policy: softmax○
- Buffer Size: 10,000→

- Batch Size: 32↓
- Gamma: 0.99→
- LR=0.0005↑
- Update every: 20↑
- HL_1: 128→
- HL_2: 64↓
- Beta: 8.0->0.01(decay=0.995)→







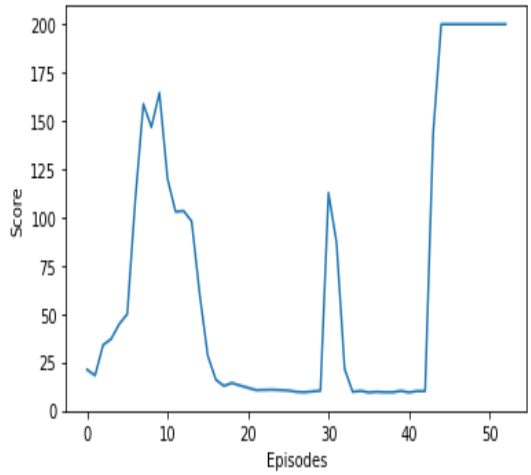
Inferences:

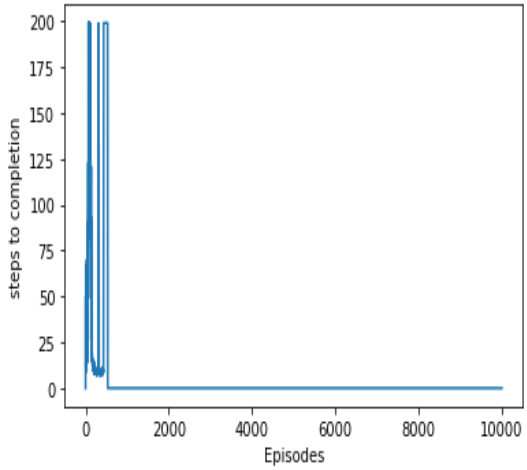
Convergence:727,806,659,1362,1971,1273,1045,884,1359,903

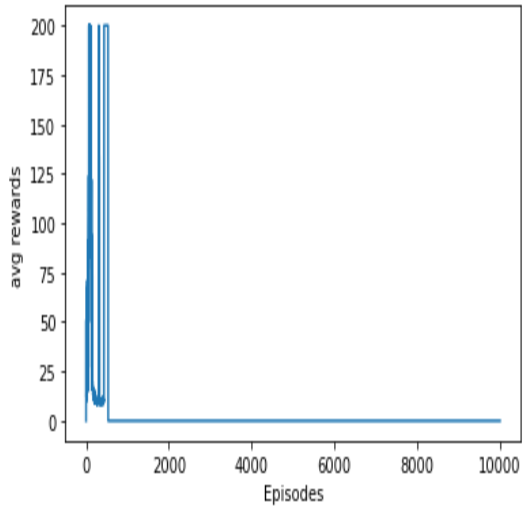
Much better performance with a lower batch size. This time softmax gave better result with respect to epsilon greedy. Looks like batch size plays a significant role for Cartpole.

5.

- Policy: softmax
- Buffer Size: 10,000→
- Batch Size: 32→
- Gamma: 0.99→
- LR=0.0005→
- Update every: 30↑
- HL_1: 128→
- HL_2: 64→
- Beta: 8.0->0.01(decay=0.995)







Inferences:

Convergence: 892,778,484,523,931,709,584,470,456,431

Best convergences and best graphs. Final evolution required a higher updation rate to give the best hyper parameters.

Summary:

For cartpole environment, once we get a stability reached by the cart we would like to repeat the same steps again to make sure that the stability of the pole is not compromised. To make sure it is repeated enough number times…like giving it chances to get its best run before a new approach is taken into account we have to increase the update every hyper parameter as increasing it gives better results. Thus we can continue the to and fro motion of the cart for the longest time until a mistake is committed and useful information is received. From the graphs we can see that the learning rate 0.0005 is the best case, as a drastic increase or a drastic decrease leads to the results getting worse. This amount is perfect for these runs. 128/128 hidden layers were complicating the learning from the experiments and the final values shows us that faster adaptation and repeating the adaptation is giving good results. So 128/64 makes sense as the second layer lessen the time to get adapted. Finally lesser batch size gives better results as the environment sends burst of feedbacks from each adaptation and taking a huge batch size will input a lot of feedback that will make the cart confused. Go simple and try simple should be applied in this environment.

B. MOUNTAIN CAR:

1. Tutorial Hyper-parameters:

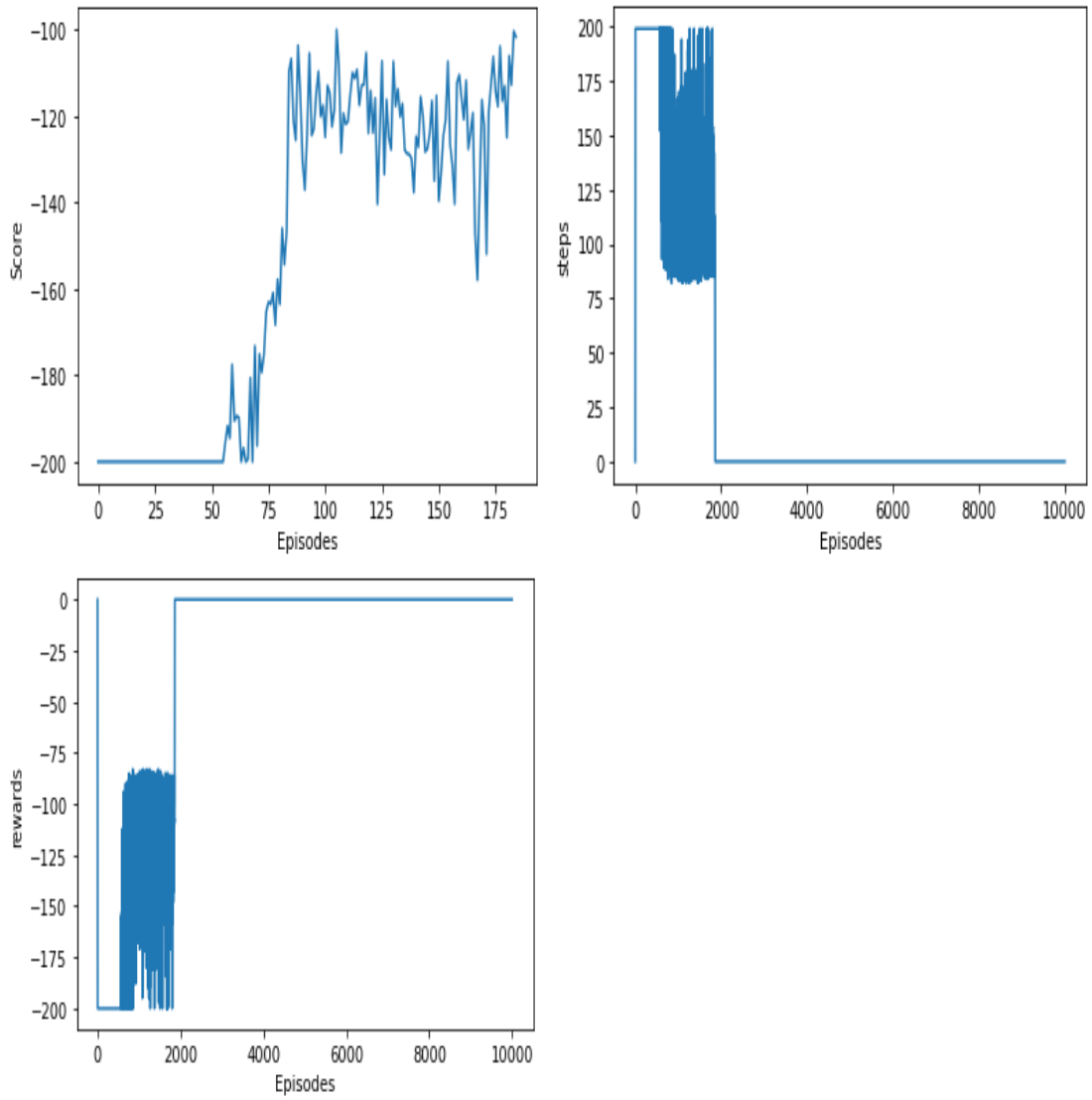
- Policy: Softmax/Epsilon Greedy
- Buffer Size: 10,000
- Batch Size: 64
- Gamma: 0.99
- LR: 0.0005
- Update every: 20
- HL_1: 128

- HL_2: 64
- Beta: 5->0.01(decay=0.995)
- Epsilon: 1->0.01(decay=0.995)

None of experiments converged for Mountain Car with the hyper-parameters given in the tutorial. So we have to start with some random values and make changes from there.

2.

- Policy: epsilon greedy
- Buffer Size: 10,000→
- Batch Size: 32↓
- Gamma: 0.99→
- LR: 0.0002↑
- Update every: 30↑
- HL_1: 64↓
- HL_2: 64↓
- Epsilon: 1.0->0.01(decay=0.9975)



Inferences:

Convergences:

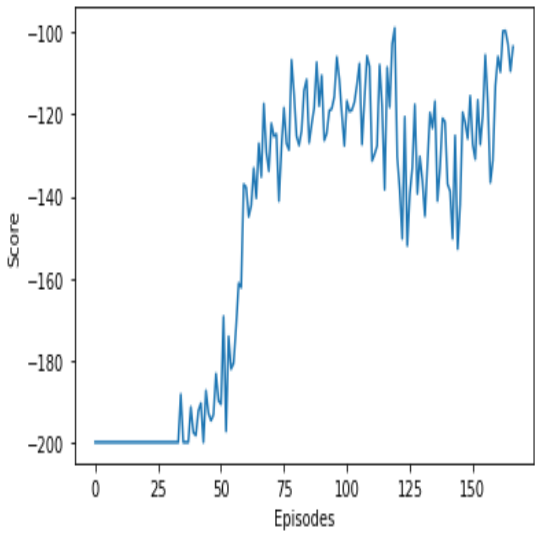
1614,1726,1219,1660,1965,2580,1659,1365,5438,1759

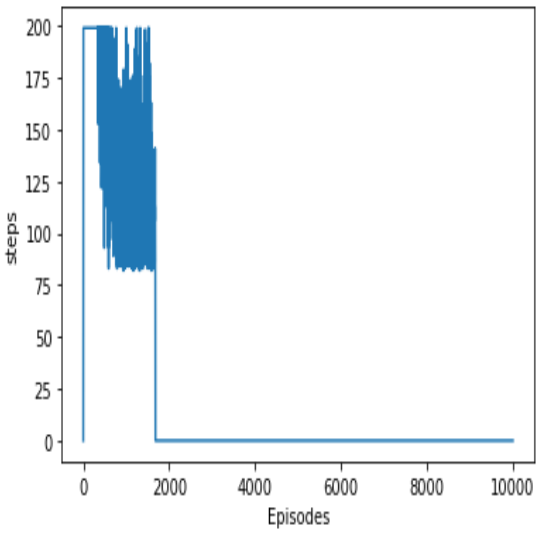
We took the extreme ends of our range of hyper-parameters that we would eventually follow. Hidden layers->64/64 of lower end, batch size 32 of lower end, Update every of 30 on the higher end. Gives a pretty decent graph with all the experiments converging on the first try. Leaves a lot to be desired too as this was just the beginning. Reaching a 3digit convergence will be our goal for now.

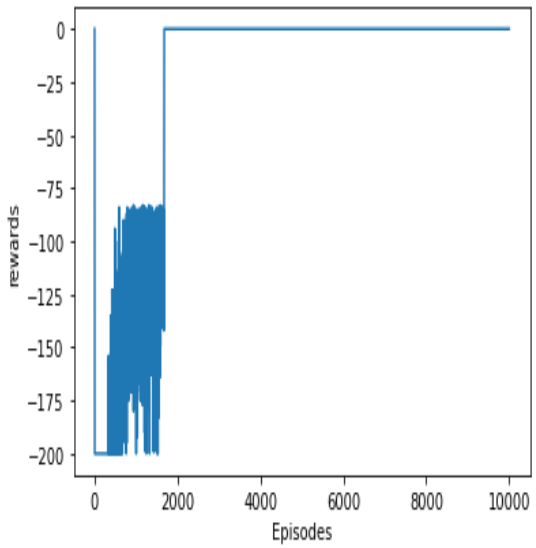
3.

- Policy: epsilon-greedy
- Buffer Size: 10,000→

- Batch Size: 64↑
- Gamma: 0.99
- LR=0.0005↑
- Update every: 50↑
- HL_1: 64→
- HL_2: 64→
- Epsilon: 1.0->0.01(decay=0.9975)







Inferences:

Convergences:

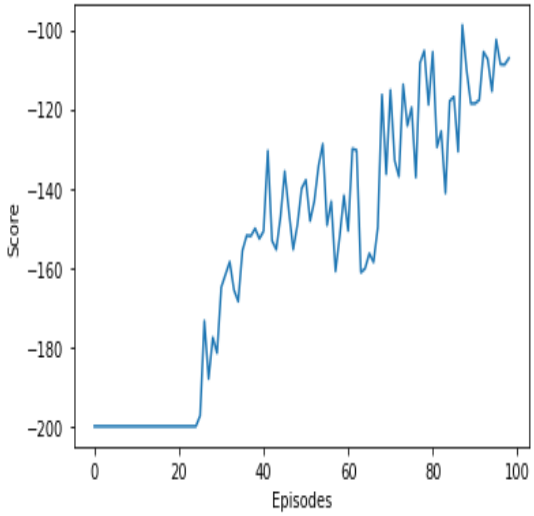
1127,1820,1975,3758,1369,1213,1415,1009,1129,1277

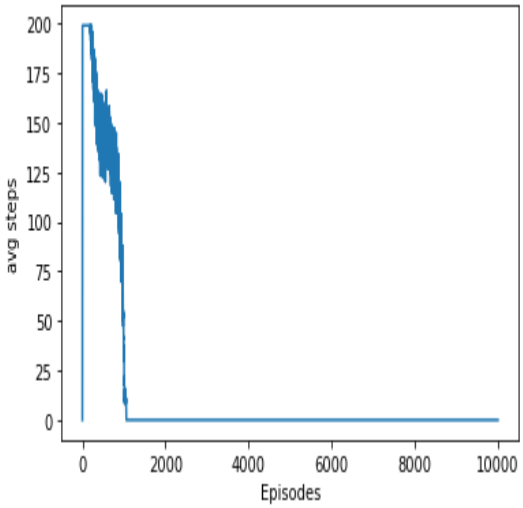
Despite a few instabilities it managed to stay grounded towards the 1000 episodes convergence. Almost the same results as before with a few increase in stabilities for now. Increase in update every didn't produce much changes. Looks like whatever influence it had was

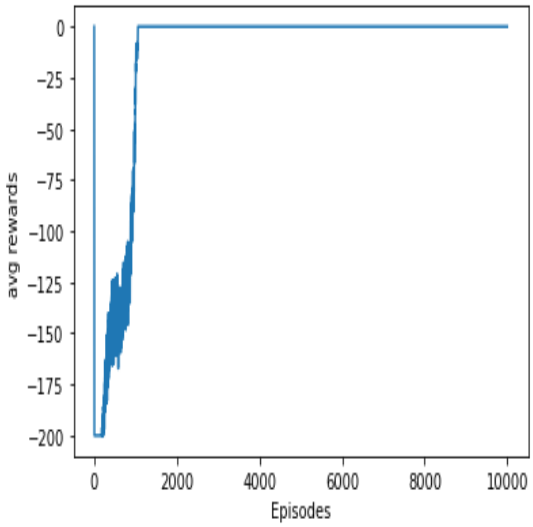
negated by another hyper-parameter, which is increase in batch size or in learning rate. Later results will show that increase in batch size had such a profound effect that it negated the effects of a higher update every hyper-parameter.

4.

- Policy: epsilon
- Buffer Size: 10,000→
- Batch Size: 64→
- Gamma: 0.99→
- LR=0.001↑↑
- Update every: 15↓↓
- HL_1: 128↑
- HL_2: 128↑
- Epsilon: 1.0->0.01(decay=0.9975)







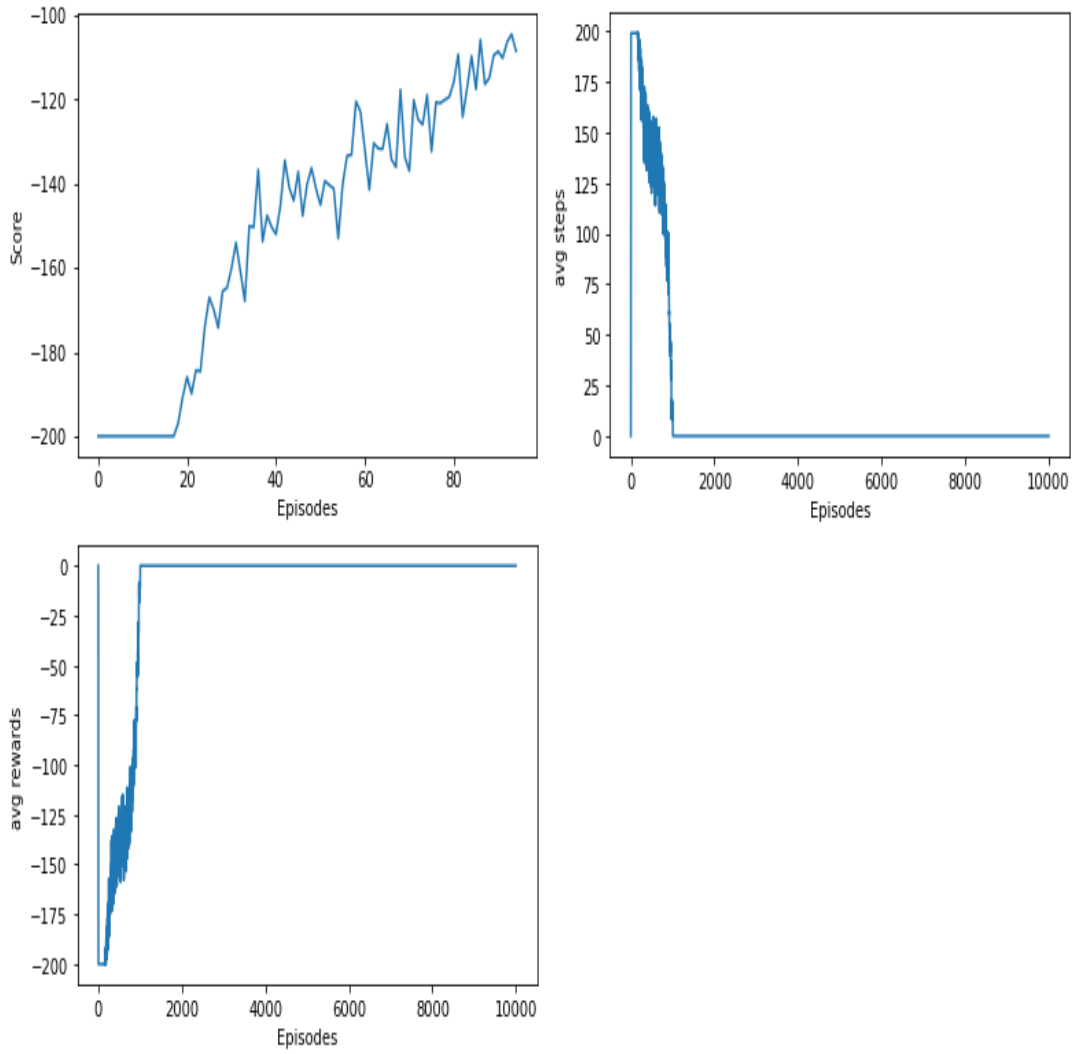
Inferences:

Convergences: 910,798,954,888,838,866,902,868,773,895

All convergences were under and around the boundary of 900 episodes. Drastically increasing learning rate and considerably reducing the update every and increasing the hidden layer sizes all had a positive effect on the agent. We will try to reduce them further.

5.

- Policy: epsilon
- Buffer Size: 10,000→
- Batch Size: 64→
- Gamma: 0.99→
- LR=0.001→
- Update every: 25↑
- HL_1: 128→
- HL_2: 128→
- Epsilon: 1.0->0.01(decay=0.9975)



Inferences:

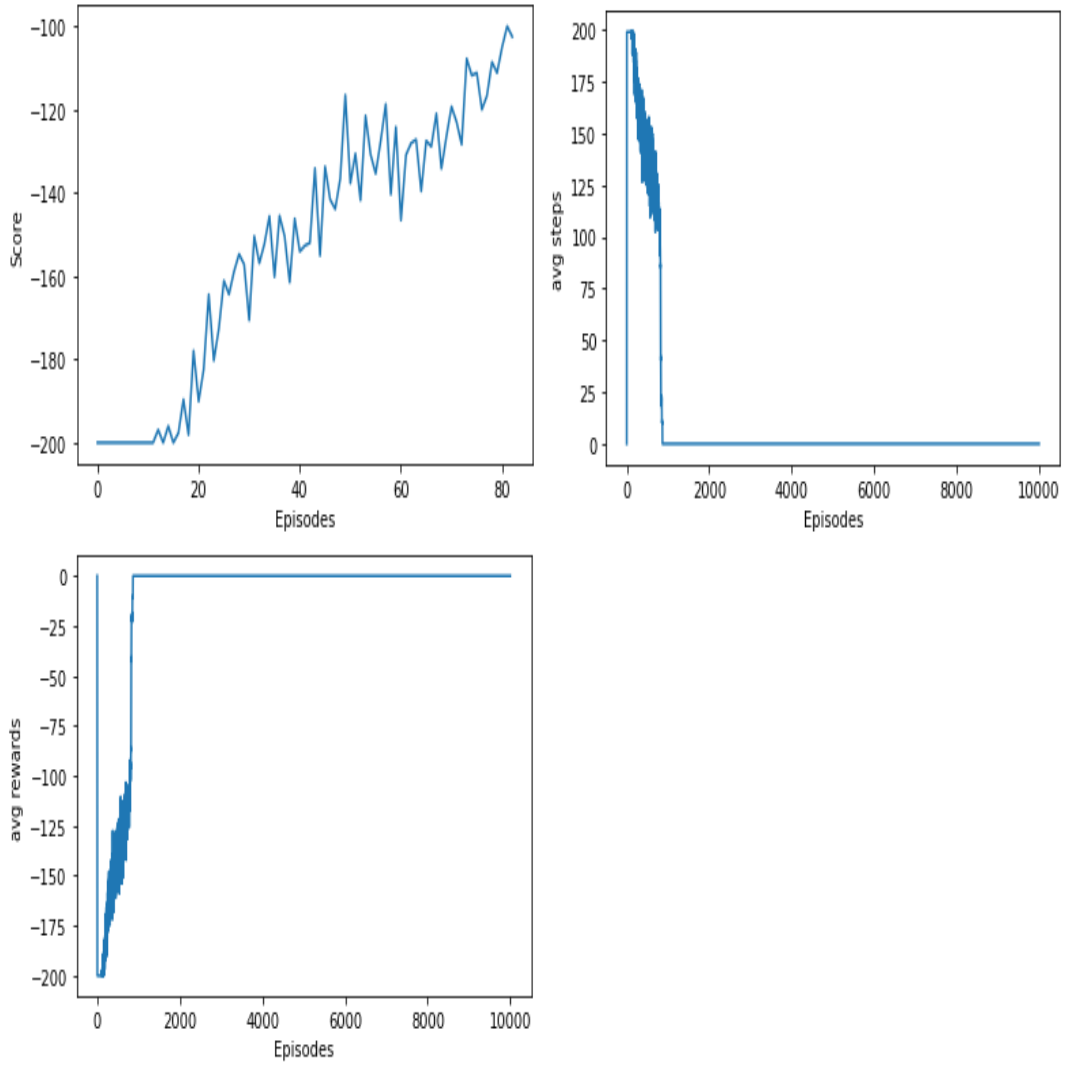
Convergences: 842,812,902,804,818,712,745,870,869,854

Average episodes required for converging were again reduced by these hyper-parameters with only increasing the update-every. After repeated testing we came to know that 25 was the threshold here…Increasing or decreasing leads to worsening.

6.

- Policy: epsilon
- Buffer Size: 10,000→
- Batch Size: 64→
- Gamma: 0.99→
- LR=0.001→
- Update every: 25→

- HL_1: 256↑
- HL_2: 256↑
- Epsilon: 1.0->0.01(decay=0.9975)



Inferences:

Convergences: 720,763,723,724,716,726,697,749,719,730

Most stable and cleanest graphs so far with convergence around 700. Just by increasing the hidden layer sizes to 256/256.

Summary: From extensive testing we found that increasing the Update every hyper-parameter or increasing it than 25 leads to changes in the average rewards drastically on the worse side. Since for Mountain car a lot of complex calculations and approximations are required higher sizes

of hidden layers often leads to increase in the stability of the graphs. The convergence boundaries get lower and stays closer to each other. With respect to cartpole the learning rate was taken much higher as the agent had to adapt much more frequently than the agent at cartpole.
Mountain car agent needs to take in a lot of feedback and adapt much more thus it needs a higher batch size that will give a lot of feedback.

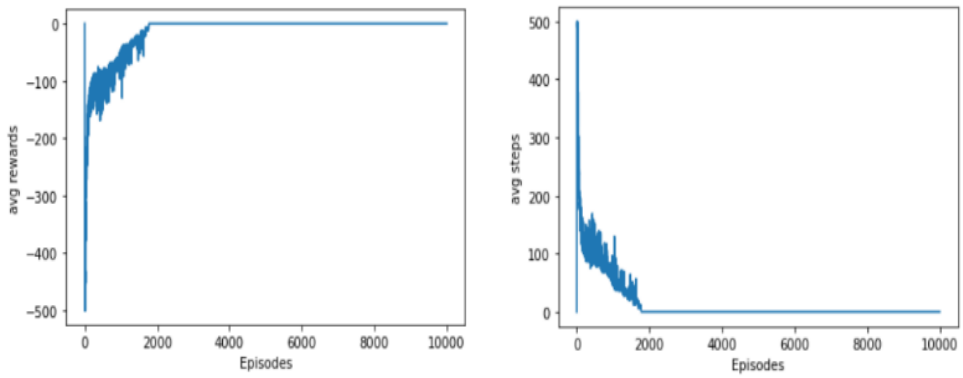
C. ACROBOT:

All experiments were done in epsilon greedy as softmax was giving really bad results. It seemed that exploitation was much more important in this environment.

1.

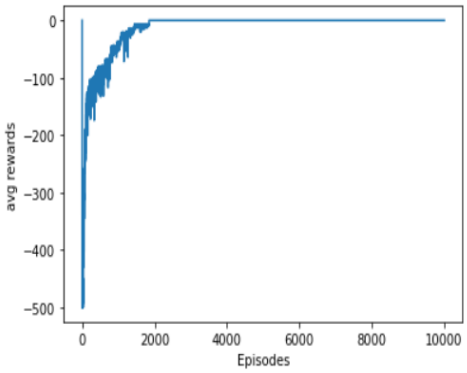
Tutorial hyperparameters were giving bad results so we took lower ends of batch size, LR, update every and higher end of hidden layer size.

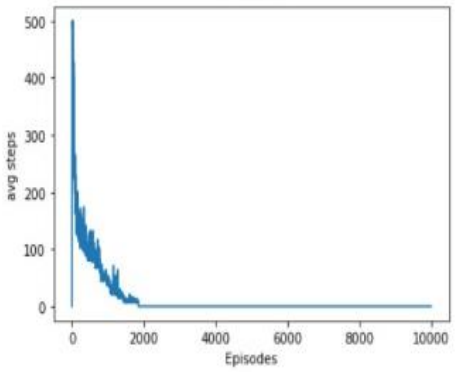
- Buffer Size = 10000
- Batch Size = 32
- Gamma = 0.99
- LR = 0.0003
- Update Every = 40
- Hidden layers = (256,256)
- Epsilon: 1.0->0.01(decay=0.9975)



2.

- Buffer Size = 10000
- Batch Size = 64
- Gamma = 0.99
- Learning Rate = 0.0002
- Update Every = 100
- Hidden layers = (256,256)
- Epsilon= 1.0->0.01(decay=0.9975)



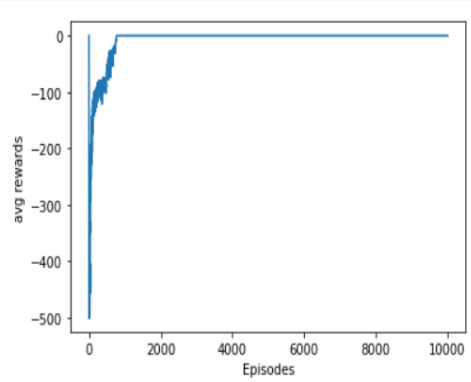


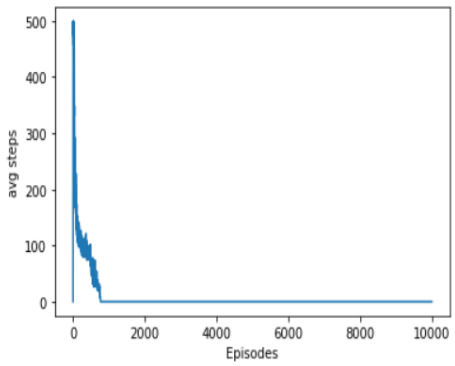
Inference:

Increasing batch size had a positive effect but wasn't represented much because of the lesser learning rate. Possibly the Update every increase played some role but whether it was positive or negative was not yet understood which will be done later.

3.

- Buffer Size = 10000
- Batch Size = 128
- Gamma = 0.99
- Learning Rate = 0.0005
- Update Every = 50
- Hidden layers = (256,512)
- Epsilon= 1.0->0.01(decay=0.9975)



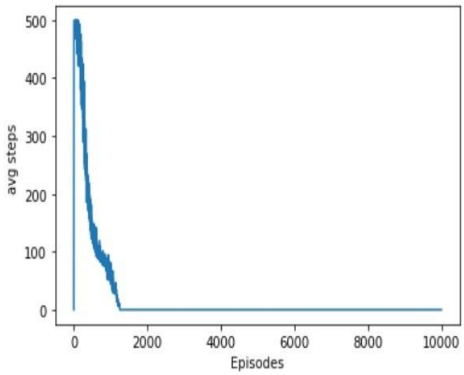


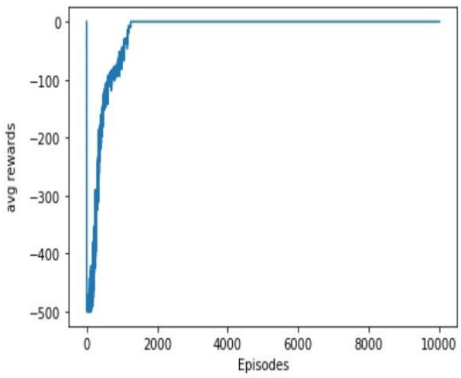
Inference:

Further increasing Batch size and learning rate(instead of decreasing it) had a really positive effect now with lower learning rate nowhere to hold them back. Update every was even lowered and if increasing it gives a positive effect then the results would have been much better. We will take a look into it in a later experiment.

4.

- Buffer Size = 10000
- Batch Size = 64
- Gamma = 0.99
- Learning Rate = 0.0004
- Update Every = 100
- Hidden layers = (256,256)





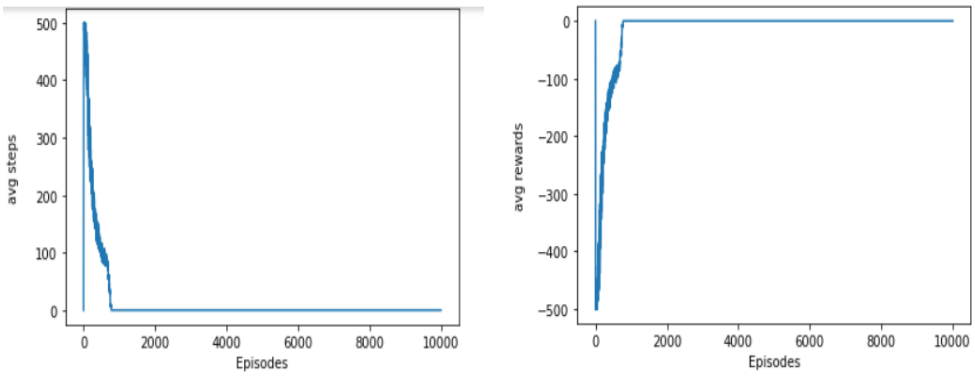
Inference:

This time batch size was decreased along with the small change in learning rate. But increasing the size of the hidden layers and the update

every had much more positive effect than last time making our theory concrete that greater update every gives better values.

5.

- Buffer Size = 10000
- Batch Size = 128
- Gamma = 0.99
- Learning Rate = 0.0005
- Update Every = 100
- First Hidden layers = (256,512)



Inferences:

So finally, as we understood from the previous experiments, we increased the batch size, the learning rate and the update every as well as the sizes of the hidden layers to get the best graphs.

Summary:

On an average epsilon greedy gave better results for the same hyper-parameters as getting the swing correct in the acrobot environment should give much more insight and increase chances of getting a higher reward again in case the same move is applied but with much more intensity. For acrobot, once we get a greater height reached by the lower end we would like to repeat the same steps again until a new height greater than the last one is not reached. So giving a high epsilon for epsilon greedy and making sure exploitation is given much more importance and tried for the most number of time permissible is the best way to go at it. To make sure it is repeated enough number times…like giving it chances to get its best run before a new approach is taken into account we have to increase the update every hyper parameter as increasing very high(100 this time w.r.t to cartpole) it gives

better results. From the graphs we can see that the learning rate 0.0005 is the best case, as a drastic increase or a drastic decrease leads to the results getting worse. This amount is perfect for these runs. 256/512 were the best ones as the complicated feed back had to be processed in this environment since it is not as simple as the cartpole environment. Since the agent needs to study complex feed back a lot of factors will be needed such as the sine, cosine of the angles made and the velocity with which the arm is swung so a greater batch size is required to produce such amount of detailed feedback for the agent to learn. So a batch size of 128 gives the best results here.

Full Step Actor Critic:

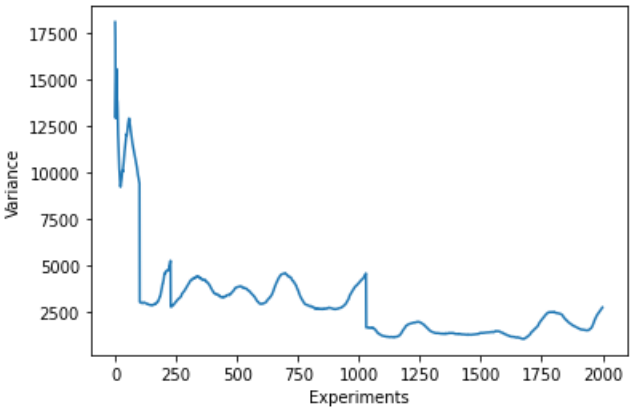
Environment: Cartpole-v1

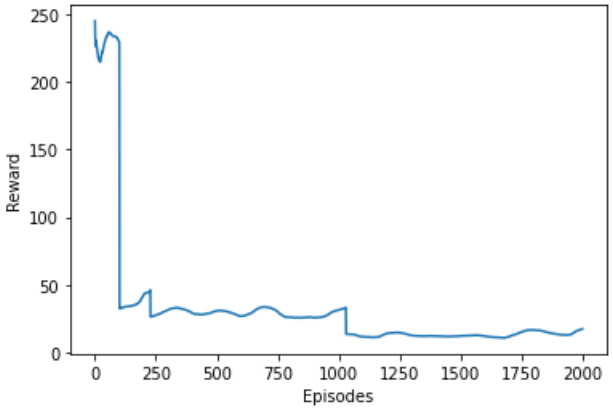
Combination 1:

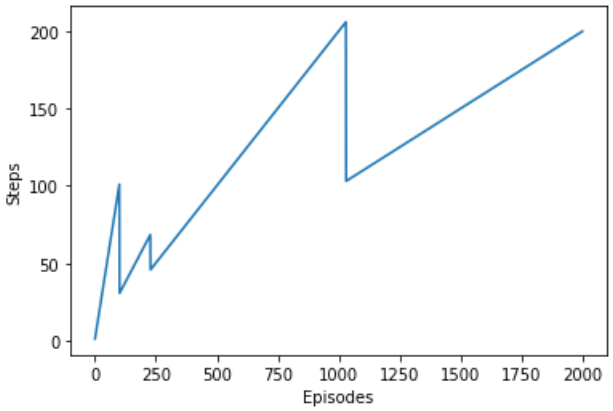
Learning Rate =0.0025

Hidden layers = 128

Gamma =0.925





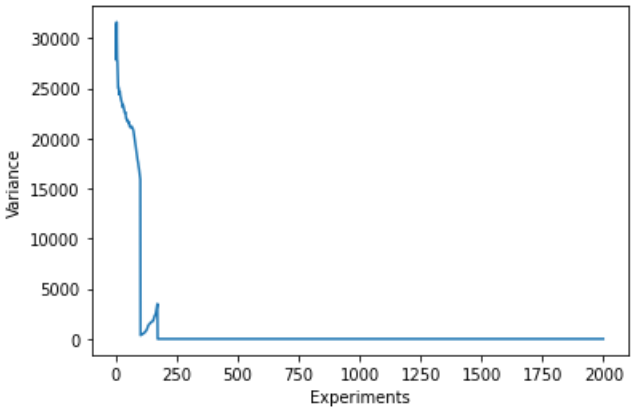


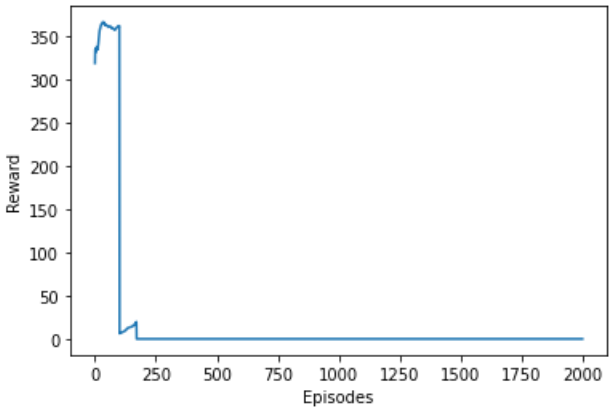
Combination 2:

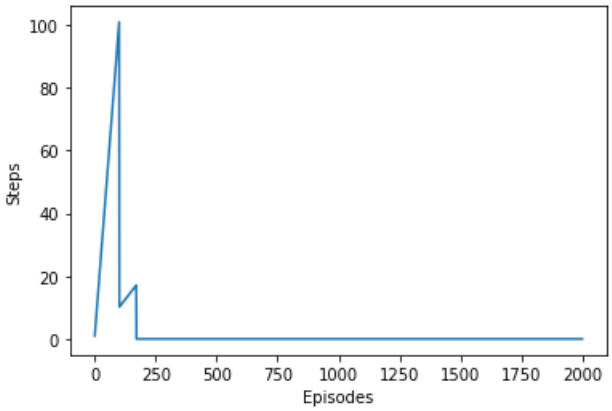
Learning Rate =0.001

Hidden layers = 256

Gamma =0.93







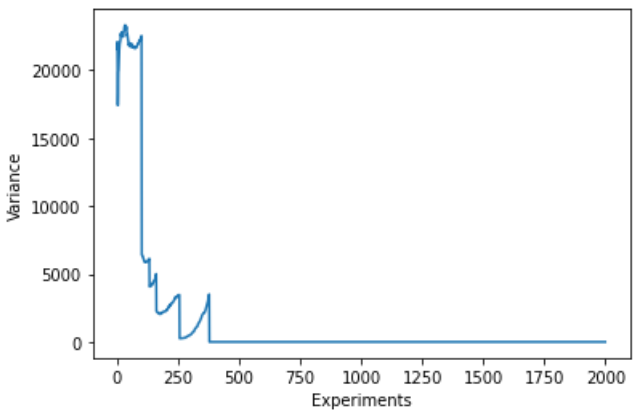
Inference: Increasing the number of hidden neurons doesn't improved the reward.

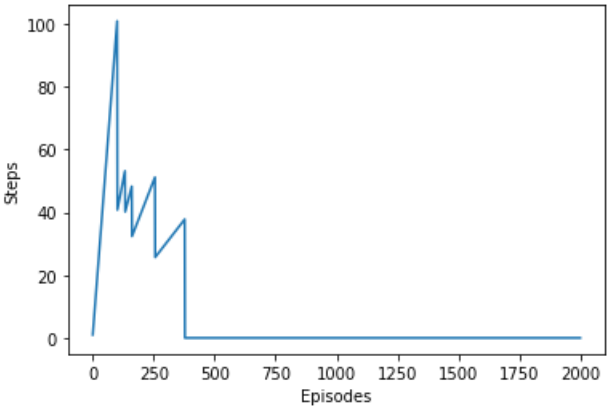
Combination 3:

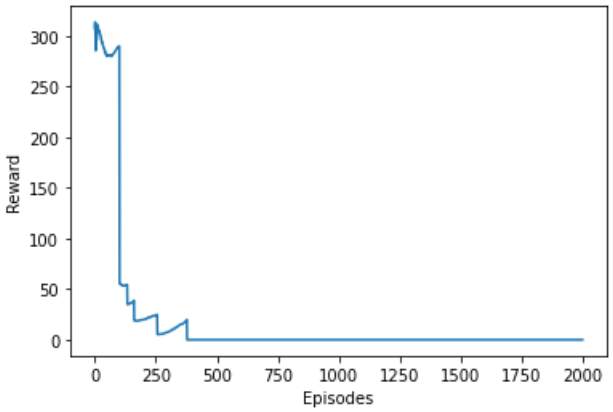
Learning Rate =0.001

Hidden layers = 128

Gamma =0.925







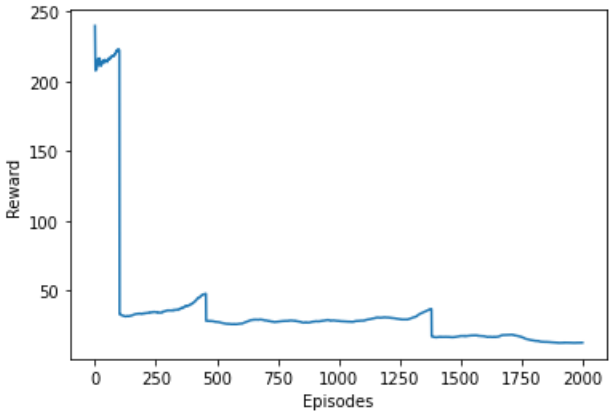
Inference: Reward is improving as gamma decreases.

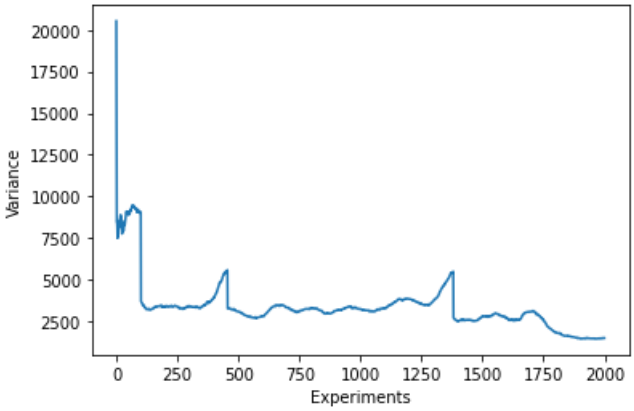
Combination 4:

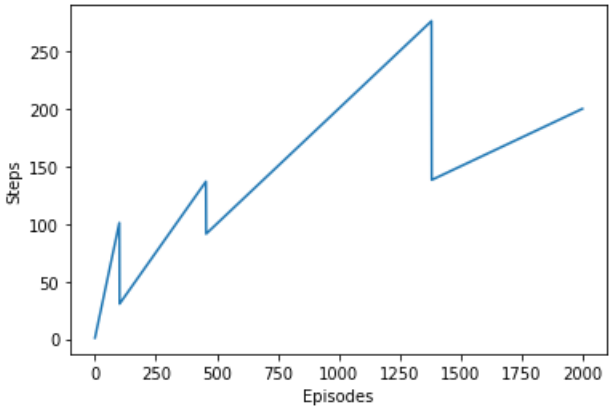
Learning Rate =0.0005

Hidden layers = 256

Gamma =0.93







Inference: Reward is again detreating while increasing the learning rate a bit.

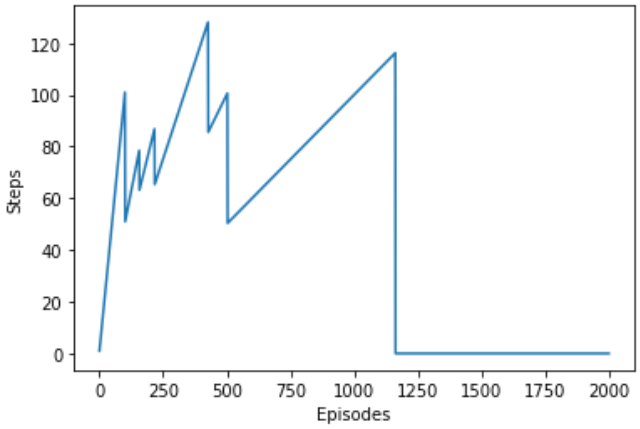
Environment: Acrobot-V1

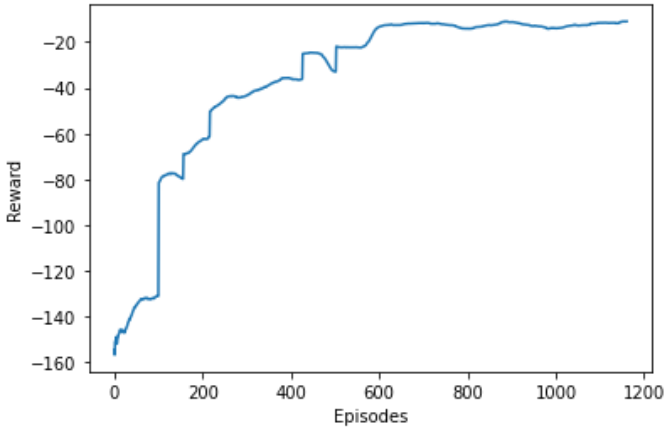
Combination 1:

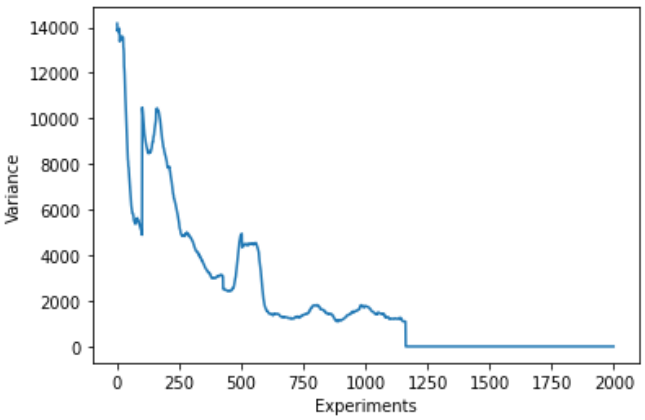
Hidden Units: 128

Learning Rate: 0.001

Gamma: 0.99





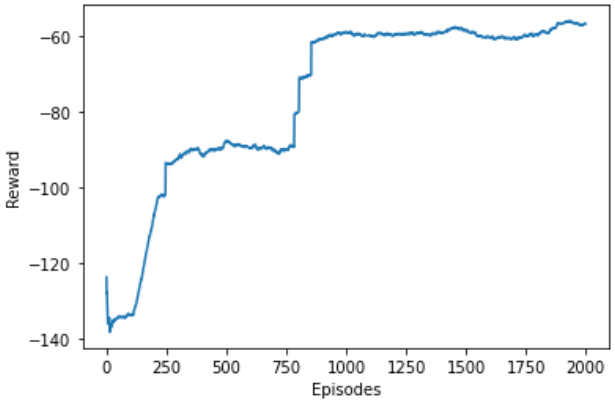


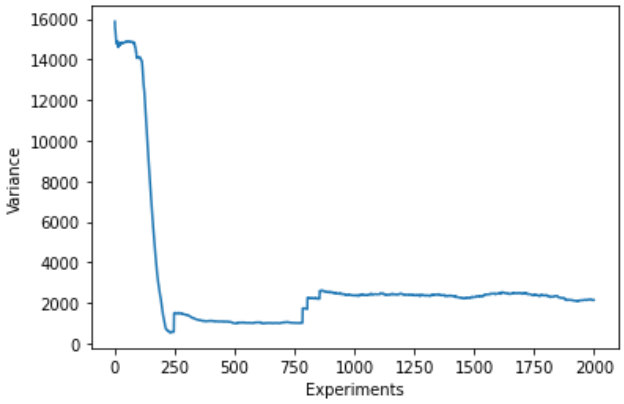
Combination 2:

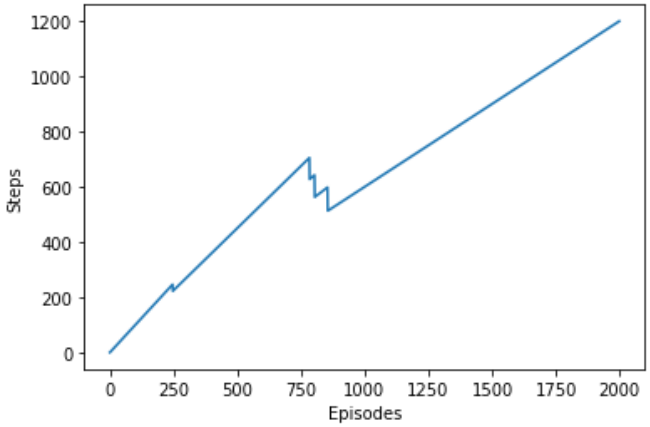
Hidden Units: 128

Learning Rate: 0.0001

Gamma: 0.99







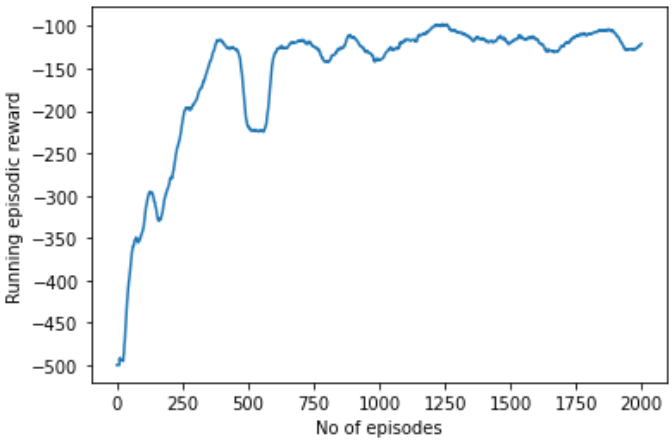
Inference: Reducing the learning rate is lowering the variance, the steps are not fluctuating much but increasing constantly as the episodes increases.

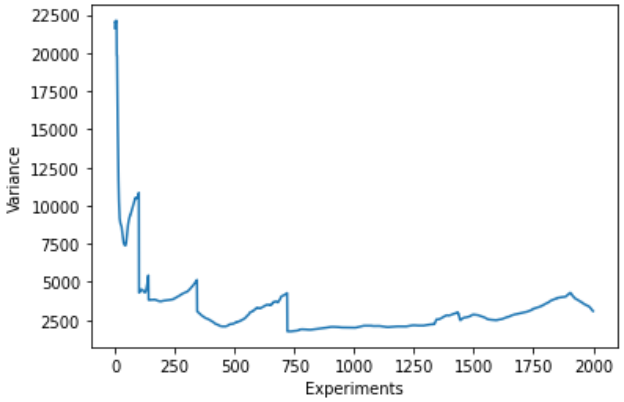
Combination 3:

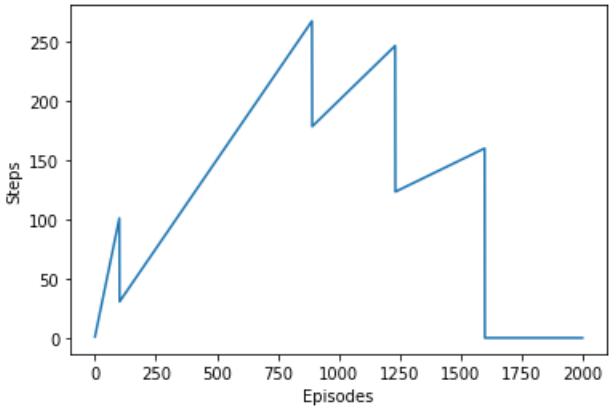
Hidden Units: 64

Learning Rate: 0.0001

Gamma: 0.99







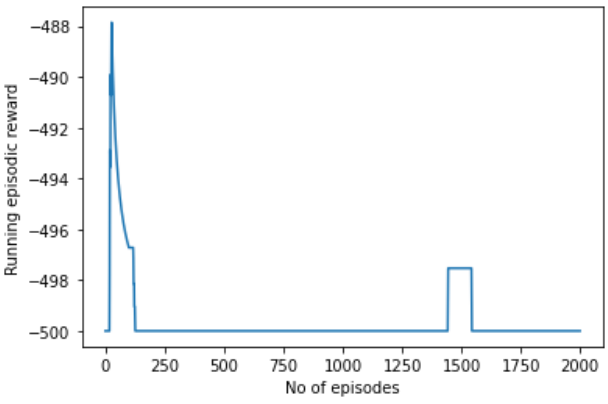
Inference: Reducing the dense layer size to 64 helped in getting a nice reward curve, variance remains the same more or less.

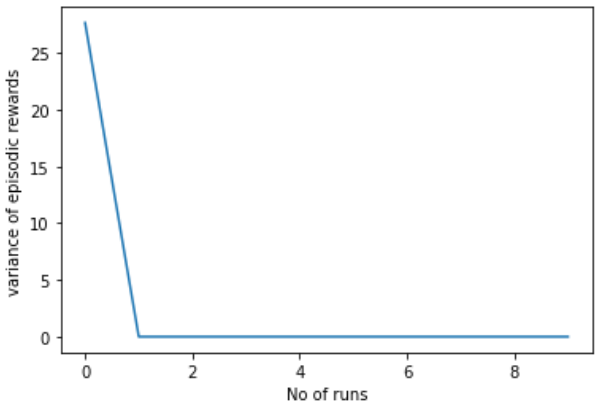
Combination 4:

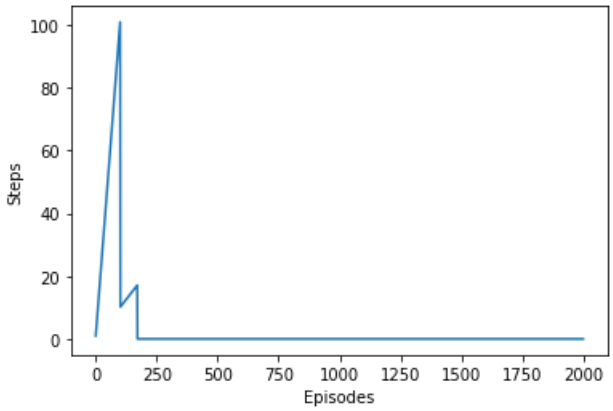
Hidden Units: 64

Learning Rate: 0.0001

Gamma: 0.99







Inference: Further increasing the learning rate giving bad rewards, steps are also increasing with the episodes.

One Step Actor Critic:

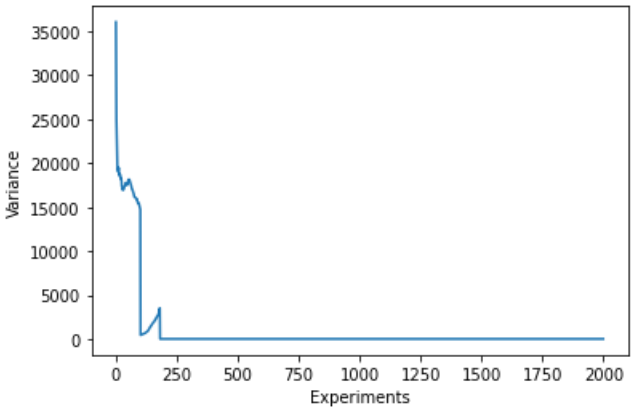
Environment: Cartpole-v1

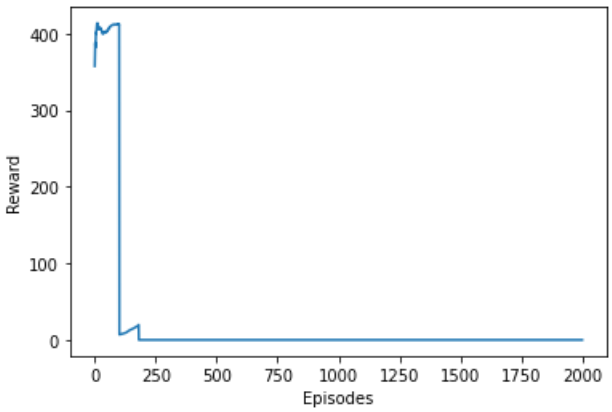
Combination 1:

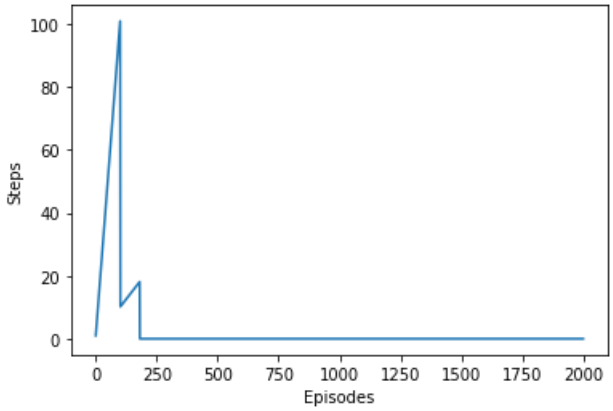
Learning Rate =0.003

Hidden layers = 256

Gamma =0.99





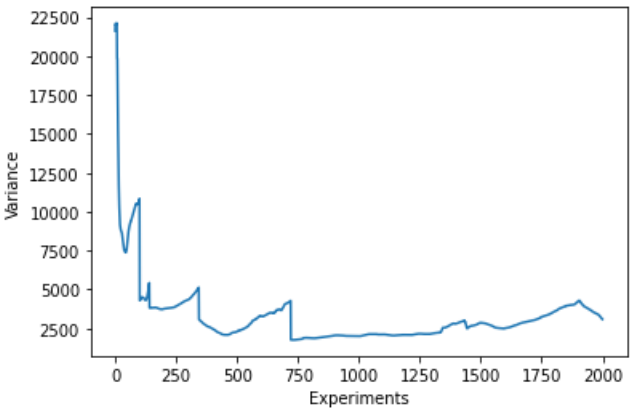


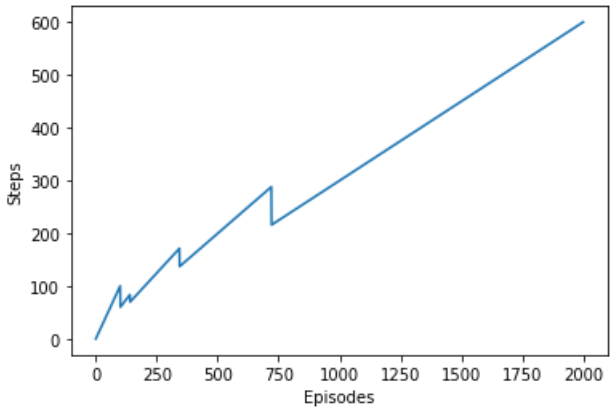
Combination 2:

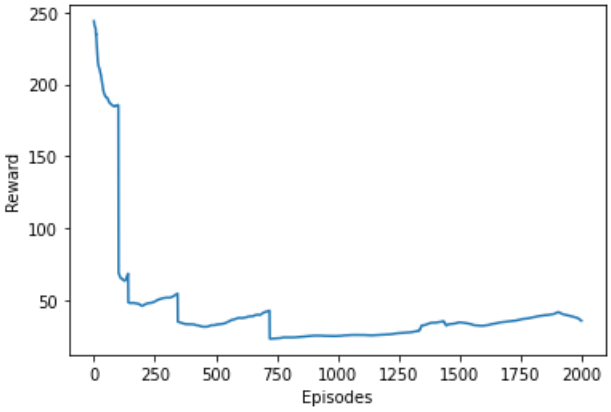
Learning Rate =0.05

Hidden layers = 256

Gamma =0.99







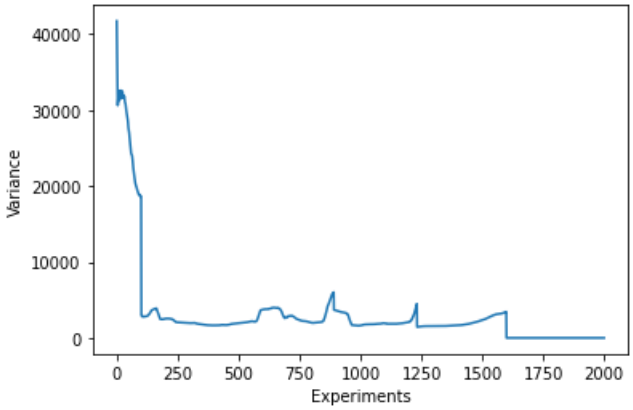
Inference: Increasing the learning rate, while keeping other parameters fixed, is negatively affecting the rewards. So, increasing learning rate is not good.

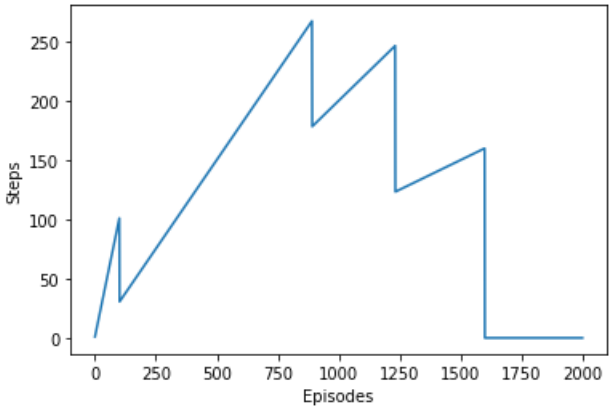
Combination 3:

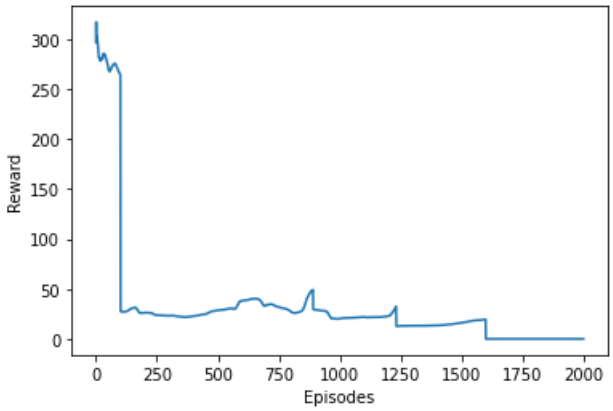
Learning Rate =0.03

Hidden layers = 128

Gamma =0.99







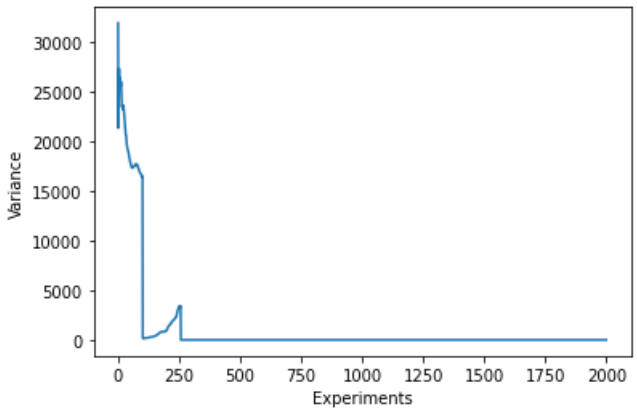
Inference: Decreasing the learning rate is improving the rewards.

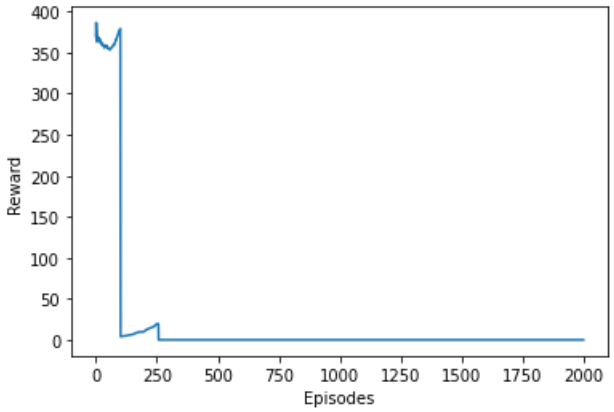
Combination 4:

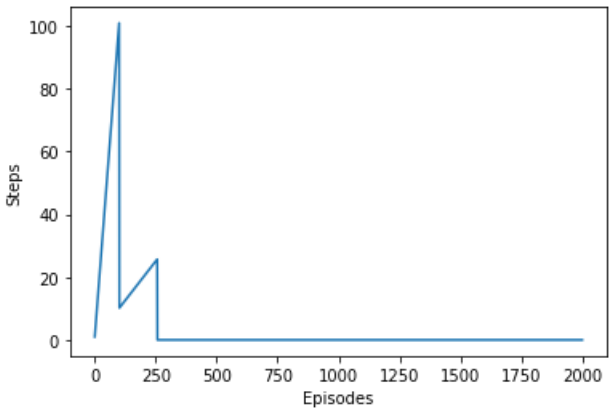
Learning Rate =0.05

Hidden layers = 32

Gamma =0.99







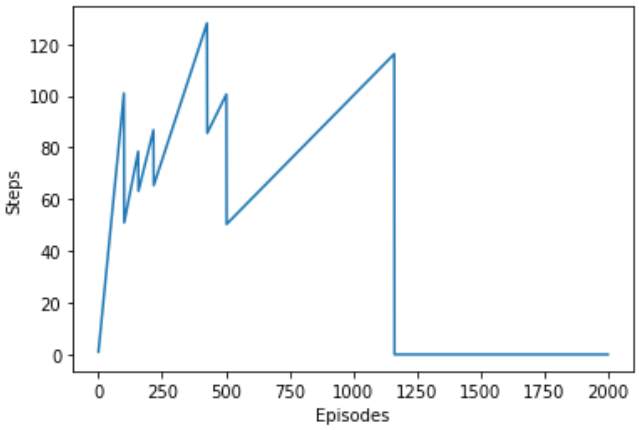
Environment: Acrobat -v1

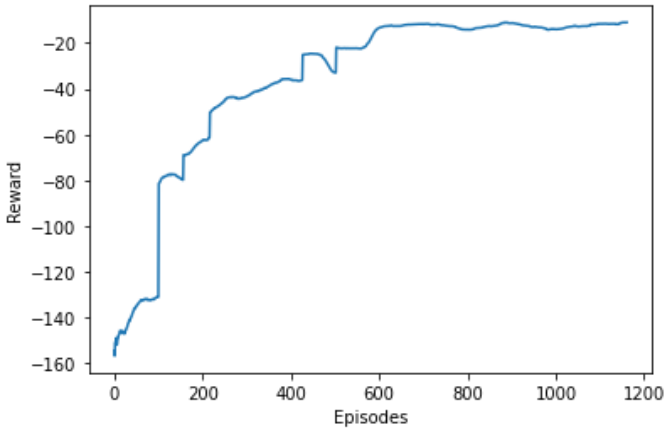
Best Hyperparameters:

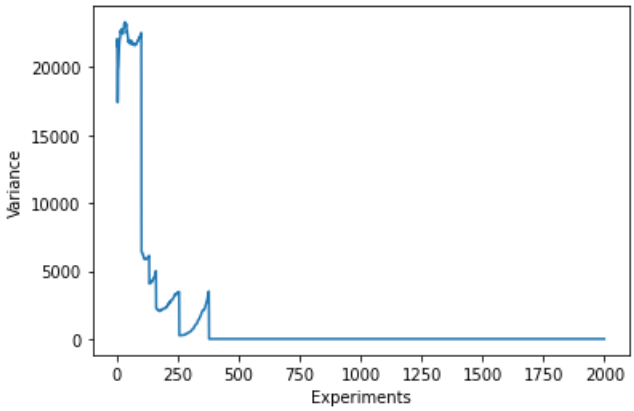
Learning Rate =0.003

Hidden layers = 256

Gamma =0.99







Conclusions:

- Variance is **highest in Single Step actor critic** and lowest in full step actor critic.
- Episode range taken to converge by the environments are:

Single step Actor Critic:

Cartpole v1: 350-400

Acrobot v1: 900-1100

MountainCar: Not at all converging and rewards are not decreasing as well.

Full step Actor Critic:

Cartpole v1: 475-700

Acrobot v1: 1300-1800

MountainCar: Not at all converging and rewards are not decreasing as well.

N step Actor Critic:

Cartpole v1: 600- 1000

Acrobot v1: 1800

MountainCar: Not at all converging and rewards are not decreasing as well.