CS6700 Reinforcement Learning
Programming Assignment-2

K V Vikram (CS19B021), Kanishkan M S (ME19B192)

March 26 2023

Introduction

This assignment familiarized us with the popular DeepRL techniques of DQNs and Actor-Critic methods. We were also introduced to the OpenAI Gym environments of **CartPole-v1**, **Acrobot-v1**, and **MountainCar-v0**.

- **Acrobot-v1:** The acrobot system includes two joints and two links, where the joint between the two links is actuated. Initially, the links are hanging downwards, and the goal is to swing the end of the lower link up to a given height.
- **CartPole-v1:** A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The system is controlled by applying a force of +1 or -1 to the cart. The pendulum starts upright, and the goal is to prevent it from falling over. A reward of +1 is provided for every timestep that the pole remains upright. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center.
- **MountainCar-v0:** A car is on a one-dimensional track, positioned between two "mountains". The goal is to drive up the mountain on the right; however, the car's engine is not strong enough to scale the mountain in a single pass. Therefore, the only way to succeed is to drive back and forth to build up momentum. All the above three environments allow only discrete actions.

DQN

Performance Metric: Average number of episodes in which the environment is solved over the five runs.

The code used for implementing DQN remains unchanged from that of Tutorial 5.

When the average reward over the last 100 episodes exceeds a reward(R$*$), we consider the environment is solved. The threshold reward(R*) for each environment is as follows:

- **Acrobot-v1:** -100
- **CartPole-v1:** 195
- **MountainCar-v0:** -125

Acrobot-v1

General Technique

For choosing the below starting hyper parameters, a few initial experimentation were done. And it was found that:

- Altering network architecture did not have a significant effect, and there was also a trade-off with the time it took for each run with increasing layers and its sizes.
- Altering truncation limit did not have any effect on the performance, tested values as low as $1 * 10^{-4}$.

Note: Instead of 10 runs, we have implemented five runs, as the experiments took a long time to run. And the results had good variations with just those five runs. (With TA's suggestion)

Reward shaping The default reward function produced good results for Acrobot environment, which is to have the free end reach a designated target height in as few steps as possible, and as such all steps that do not reach the goal incur a reward of -1. Achieving the target height results in termination with a reward of 0.

Starting Hyperparameters

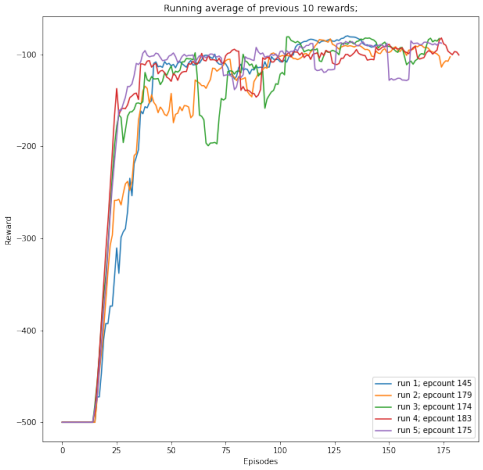
- hidden layers = 2
- hidden sizes $[64, 128]$
- truncation limit = $1 * 10^{32}$
- learning rate(α) = $5 * 10^{-4}$
- discount factor (γ) = 0.99
- buffer size = 40000
- batch size = 64
- update every = 100
- exploration policy = epsilon-greedy
- decay factor = 0.8
- epsilon start = 20
- epsilon end = $1 * 10^{-4}$

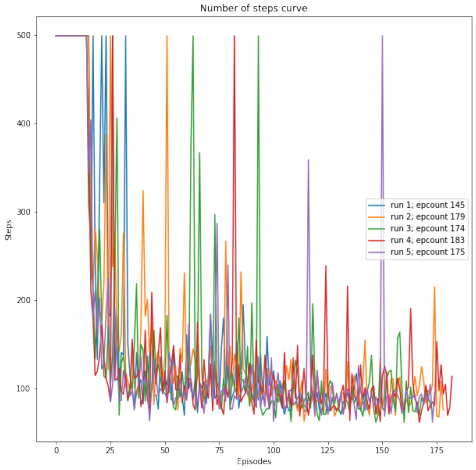
Experiment 1

Current Hyperparameters

- Here, we use the starting hyperparameters to perform the experiment.

Reward Curve and Steps Curve for the 5 runs





Justification

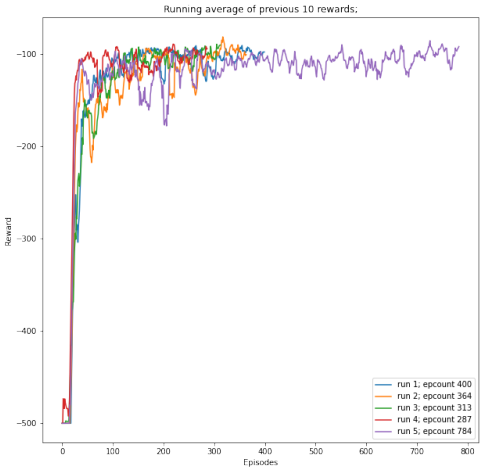
- From the reward and steps curve, it is evidently visible that the agent is stabilizing around the threshold reward for all the runs for our choice of starting hyperparameters.
- We came up with these default parameters after tweaking and experimenting around for some iterations.

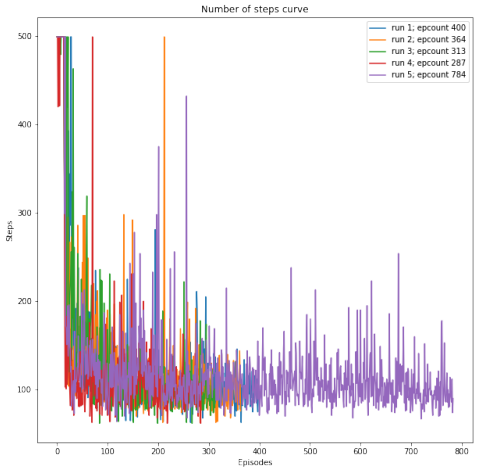
Experiment 2

Current Hyperparameters

- *Discount factor*(γ) is changed from 0.99 to 0.97 and tested for this experiment
- There is only one change from the starting hyperparameters.

Reward Curve and Steps Curve for the 5 runs





Justification

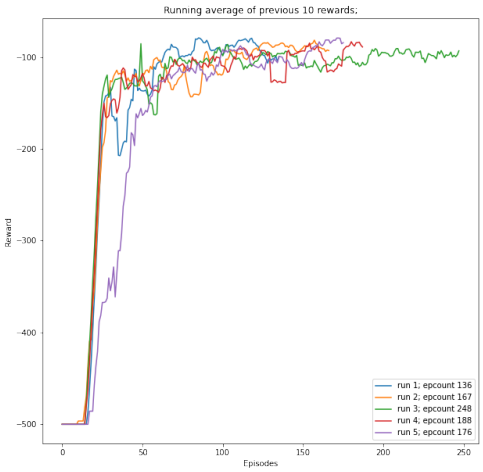
- In this experiment, there is an overall increase in the number of episodes before reaching the threshold reward with the maximum episode count being 784 in the fifth run.
- This is due to the change in discount factor by 0.02, from 0.99 to 0.97. This is because the lesser discount factor led the agent to focus on immediate rewards than long-term rewards.

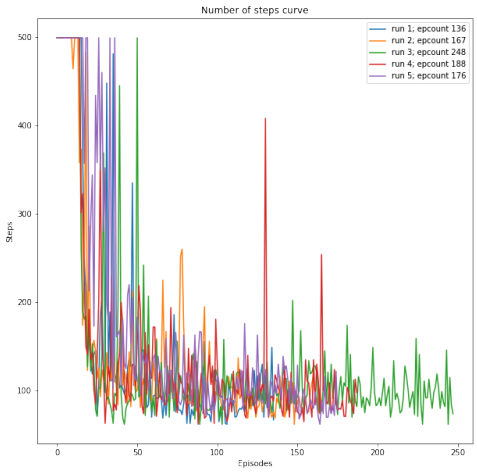
Experiment 3

Current Hyperparameters

- *Discount factor* resorted back to the original value(0.99), since experiment 2 did not show improved performance
- *Learning rate* changed to $9 * 10^{-4}$
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

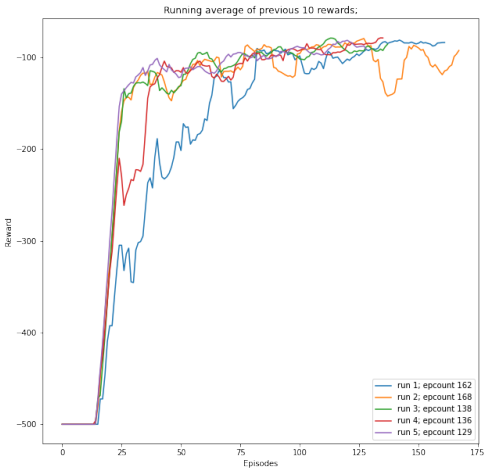
- There is a slight increase in the episode count in this experiment, but much less than the increase in 2nd experiment.
- Thus it shows that increasing the learning rate did not help the agent to learn, but rather made it bounce around the optimal policy.

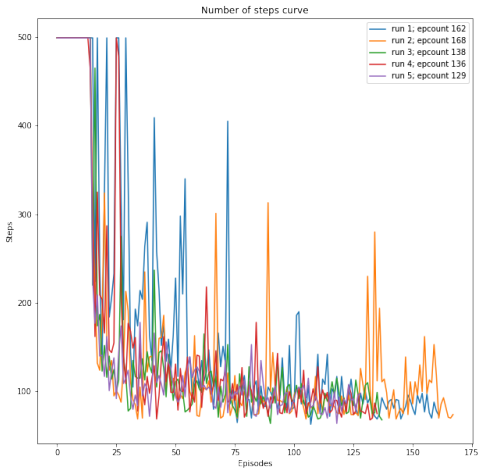
Experiment 4

Current Hyperparameters

- *Learning rate* resorted back to the original value($5 * 10^{-4}$) since it did not improve the performance
- *Buffer size* decreased to 10000
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

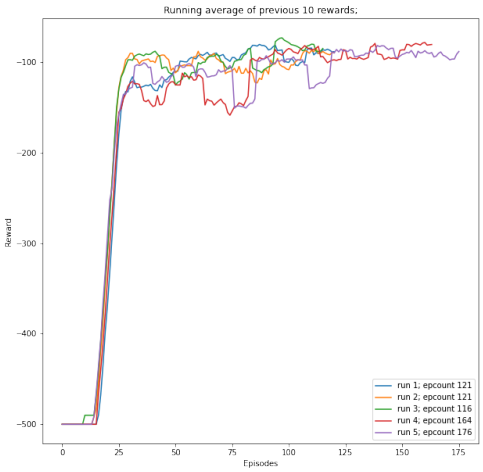
- There is a considerable increase in the performance(decrease in the episode count) in this experiment, after reducing the buffer size to 10000.
- Smaller buffer size contains more recent experiences and this improvement in performance shows that for Acrobot environment recent information is more relevant

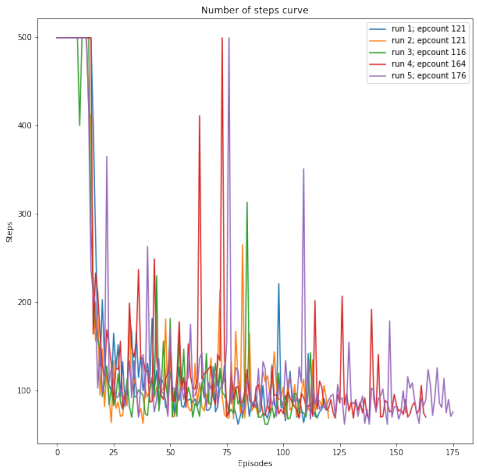
Experiment 5

Current Hyperparameters

- Decreased *Buffer size* from the previous experiment improved the performance. Therefore this parameter is fixed
- *Batch size* increased to 128
- Now there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

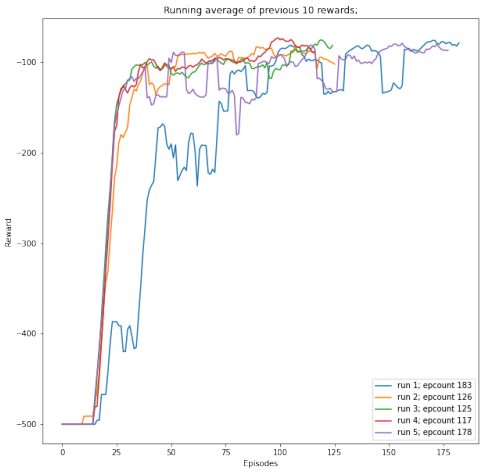
- There is a considerable increase in the performance(decrease in the episode count) in this experiment after increasing the batch size to 128.
- Larger batch size has helped reduce the variance in gradient estimates leading to better performance.

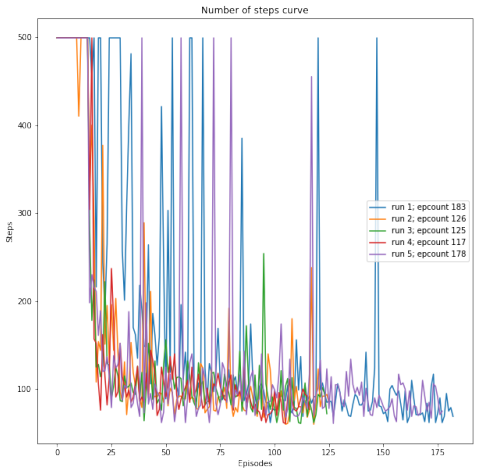
Experiment 6

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* increased to 256
- Now there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

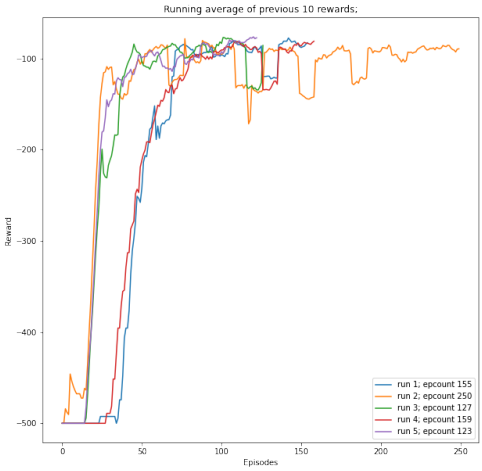
- Increase in batch size should have given better performance even though it took more time to converge
- But there is a dip in the performance(increase in the episode count) in this experiment, after increasing batch size further to 256

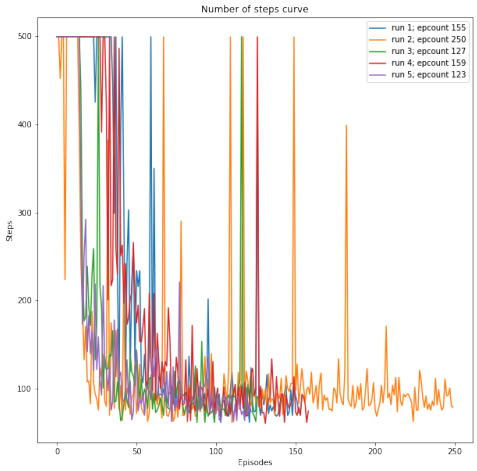
Experiment 7

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* is now fixed to 128 on comparing the performances from Experiments 4,5 and 6
- *Update every* is now varied to a value of 20
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

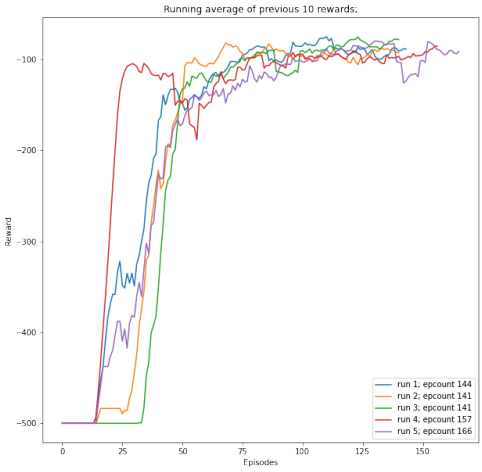
- Performance reduced a bit after changing the update frequency down to 20
- Reducing the update frequency might have slowed down the convergence, but the performance could have actually increased if other hyperparameters were tuned accordingly
- For this set of starting hyperparameters, reducing update frequency does not help in increasing performance

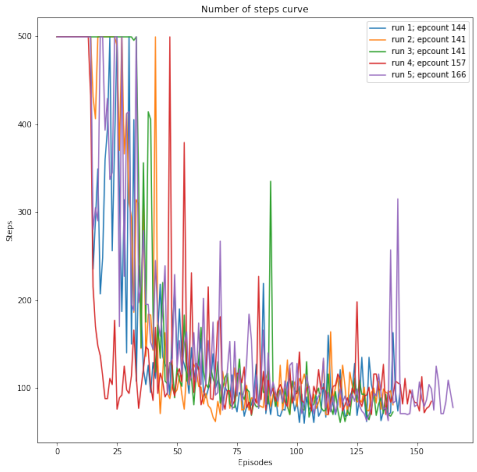
Experiment 8

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* is now varied to a value of 50
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

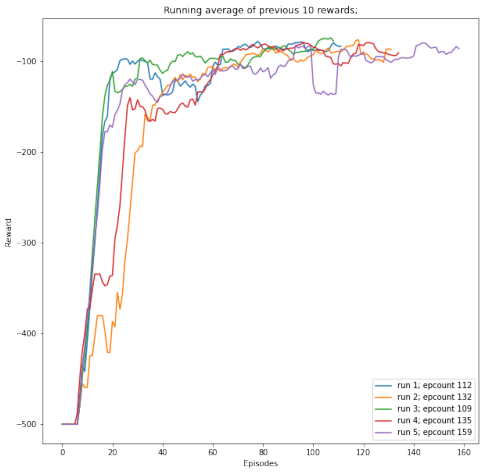
- For an update frequency of 50, performance is comparatively better than when the update frequency was 20.
- But not as good as the performance when the update frequency is 100
- For 100, the convergence was fast enough with lower variance in Q values

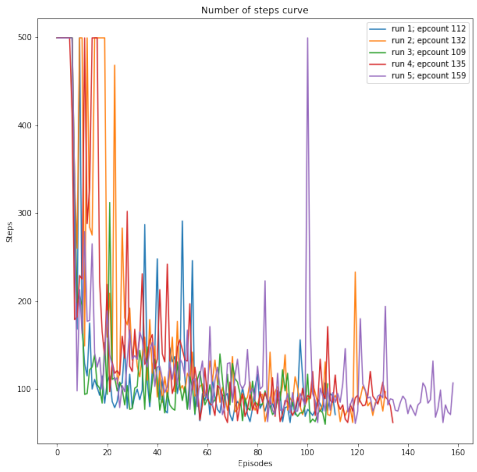
Experiment 9

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* will remain 100(Starting parameter), as varying it in Experiments 7 and 8 did not improve the performance
- *Decay factor* is now varied to a value of 0.6
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

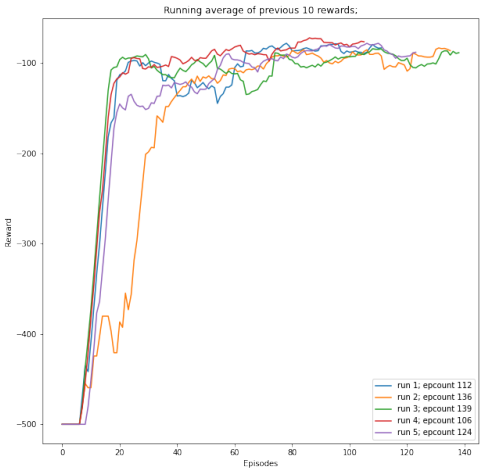
- Performance increased significantly on decreasing decay factor from 0.8 to 0.6
- This shows that the agent needed more exploration time to figure out the optimal policy, which it was not able to do till now with a 0.8 decay factor, and it was performing sub-optimal actions

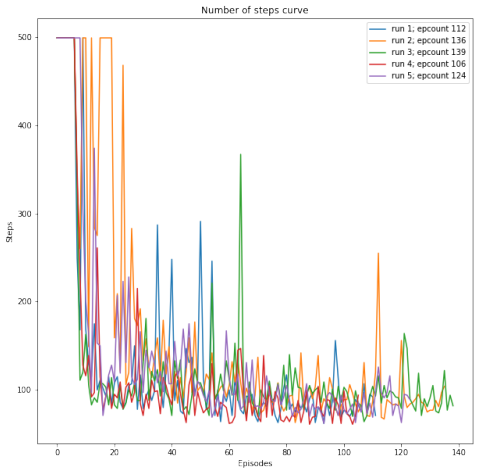
Experiment 10

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Decay factor* of value 0.6 improved the performance of the model
- *Epsilon end* is varied to $1 * 10^{-5}$
- Now there are four changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

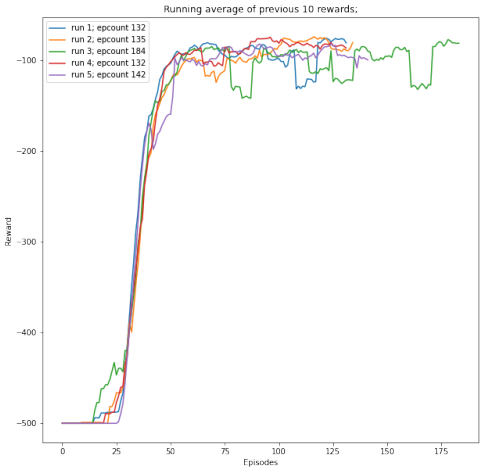
- There was an even better improvement in the agent learning after decreasing the epsilon end value to $1 * 10^{-5}$
- This shows that after a considerable amount of exploration, decreased epsilon end value helped the policy to exploit the best possible actions

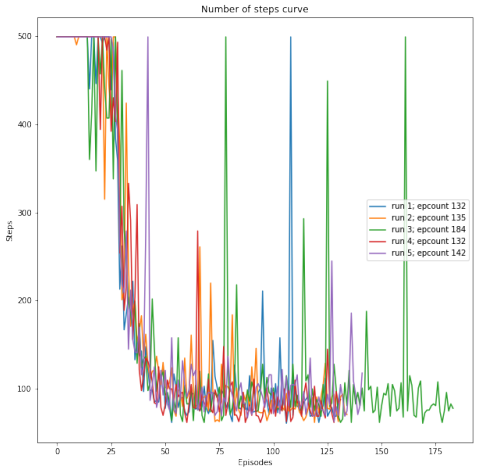
Experiment 11

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Decay factor* is 0.9 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are six changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

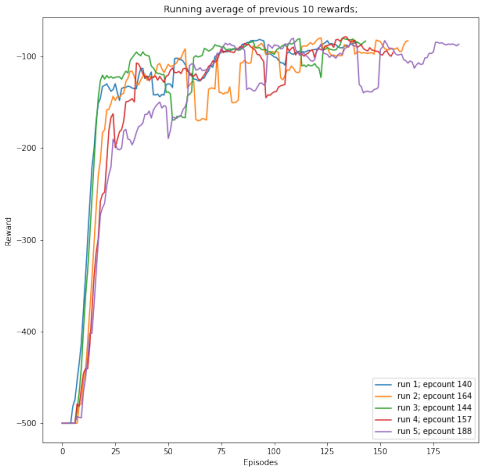
- Softmax exploration policy produced a very good results, but not as good as the 10th experiment result.
- It is also possible that the hyperparameter choices that we took for the softmax strategy were not good enough that it performed sub-optimally.

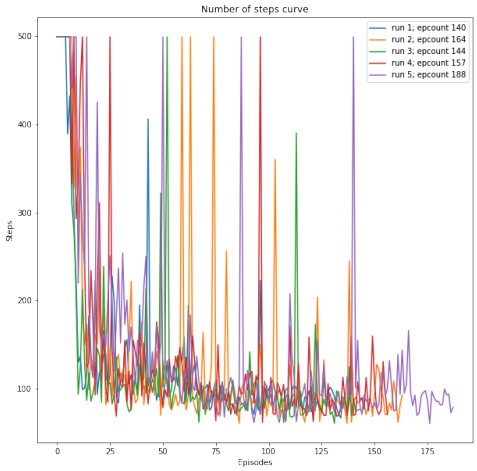
Experiment 12

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Decay factor* is 0.6 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are six changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

- Performance is much better than experiment 11 due to the low decay factor this time.
- But it did not outperform epsilon greedy's best configuration

Best configuration for Acrobot-v1 environment

Out of all the experiments, the 10th experiment had the best performance of 123.4, whereas starting parameters had a performance of 171.2

CartPole-v1

General Technique

For choosing the below-starting hyperparameters, a few initial experimentation were done. And it was found that:

- Altering network architecture did not have a significant effect, and there was also a trade-off in the time it took for each run with increasing layers and its sizes.
- Altering truncation limit did not have any effect on the performance, tested values as low as $1 * 10^{-4}$
- Discount factor was also not altered from 0.99 as it did not produce better performance

Note: Instead of 10 runs, we have implemented five runs, as the experiments took a long time to run. And the results had good variations with just those five runs.

Reward shaping The default reward function was changed for this environment for better and faster reward convergence. The custom reward function used is as follows:

```
def reward_shaping(state, next_state):
return -100*(np.abs(next_state[2])- np.abs(state[2]))
```

The default reward is changed to a value proportional to the decrease in the absolute value of the pole angle. This way it gets rewarded for actions that bring the pole closer to the equilibrium position.

Starting Hyperparameters

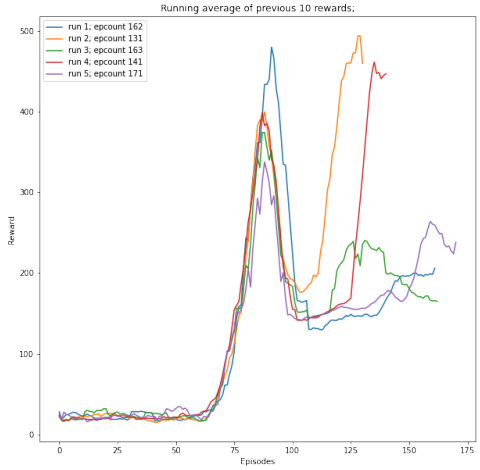
- hidden layers = 2
- hidden sizes $[64, 128]$
- truncation limit = $1 * 10^{32}$
- learning rate(α) = $5 * 10^{-4}$
- discount factor (γ) = 0.99
- buffer size = 40000
- batch size = 128
- update every = 100
- exploration policy = epsilon-greedy
- decay factor = 0.95
- epsilon start = 20
- epsilon end = $1 * 10^{-4}$

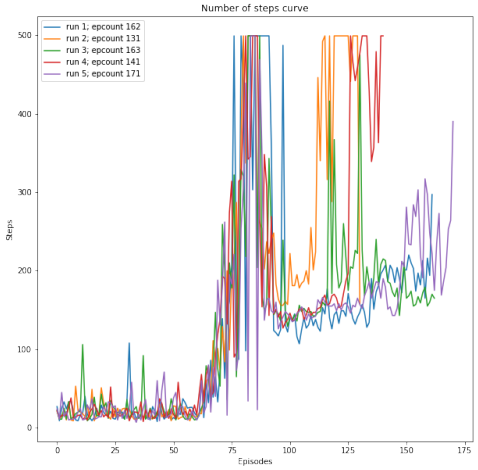
Experiment 1

Current Hyperparameters

- Here, we use the starting hyperparameters to perform the experiment.

Reward Curve and Steps Curve for the 5 runs





Justification

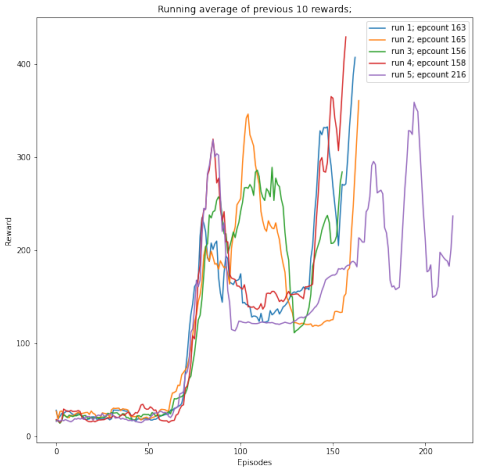
- From the reward and steps curve,it is evidently visible that the agent is stabilizing around the threshold reward for all the runs for our choice of starting hyperparameters
- We came up with these default parameters after tweaking and experimenting around for some iterations.

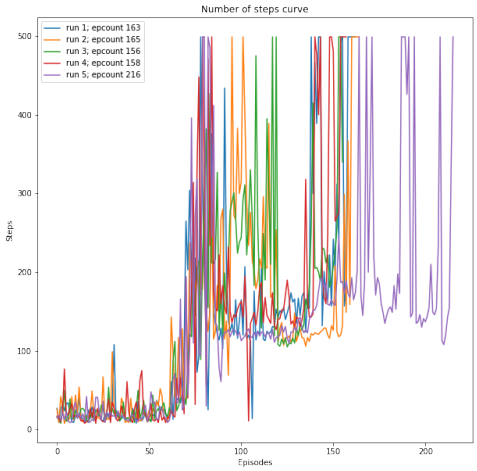
Experiment 2

Current Hyperparameters

- *Learning rate* changed to $9 * 10^{-4}$
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

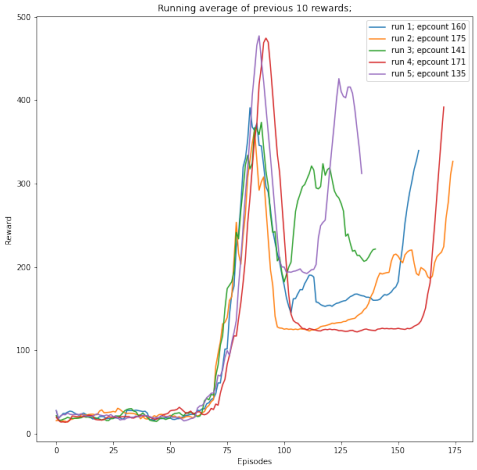
- There is a slight increase in the episode count in this experiment, but much less than the increase in 2nd experiment
- Thus it shows that increasing the learning rate did not help the agent to learn, but rather made it bounce around the optimal policy.

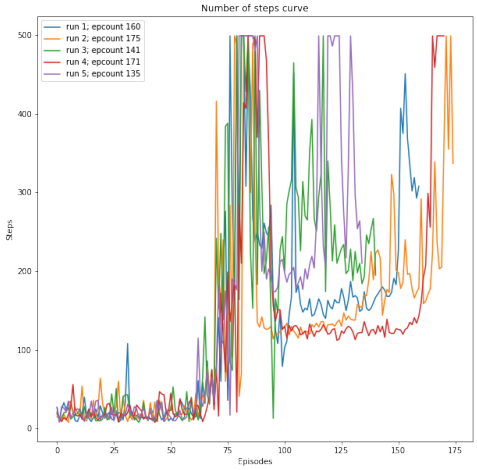
Experiment 3

Current Hyperparameters

- *Learning rate* resorted back to the original value($5 * 10^{-4}$) , since experiment 2 did not show improved performance
- *Buffer size* increased to 80000
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

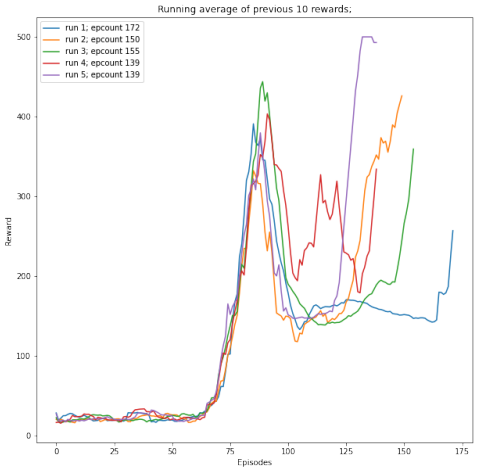
- There is a decrease in the performance(increase in the episode count) in this experiment, after increasing the buffer size to 80000
- Higher buffer size contains more experiences in the past, and this decline in performance shows that for the CartPole environment recent information is more relevant

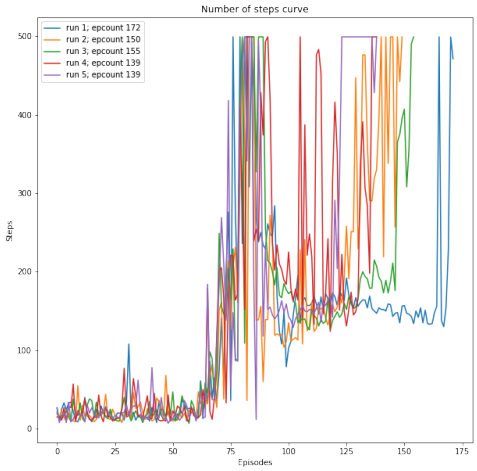
Experiment 4

Current Hyperparameters

- *Buffer size* decreased to 10000
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

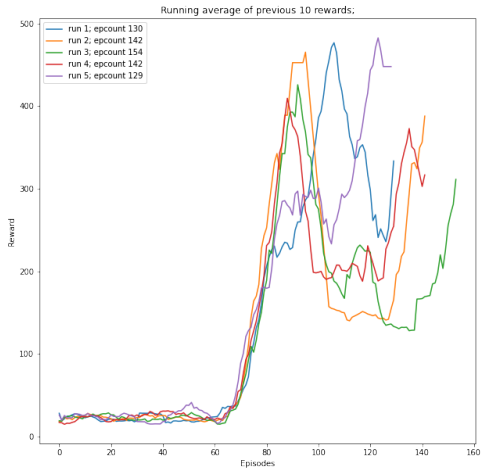
- There is a considerable increase in the performance(decrease in the episode count) in this experiment, after reducing the buffer size to 10000
- Smaller buffer size contains more recent experiences and this improvement in performance shows that for Acrobot environment recent information is more relevant

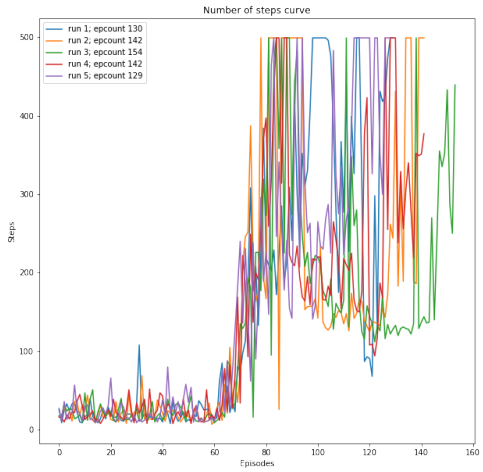
Experiment 5

Current Hyperparameters

- Decreased *Buffer size* from the previous experiment improved the performance. Therefore this parameter is fixed
- *Batch size* decreased to 64
- Now there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

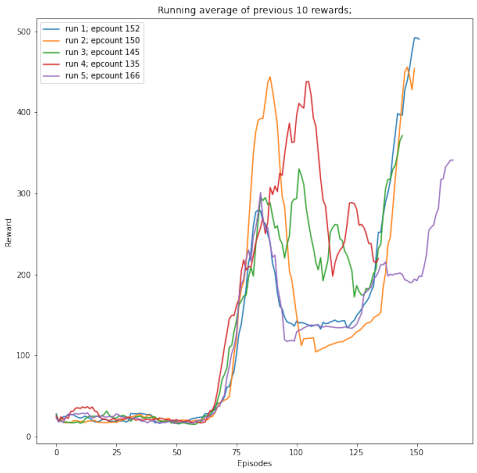
- Reducing batch size from 128 to 64 has improved the performance considerably
- This is because smaller batches provide more frequent updates to the network, allowing it to learn faster and more accurately from the experiences in the replay buffer.

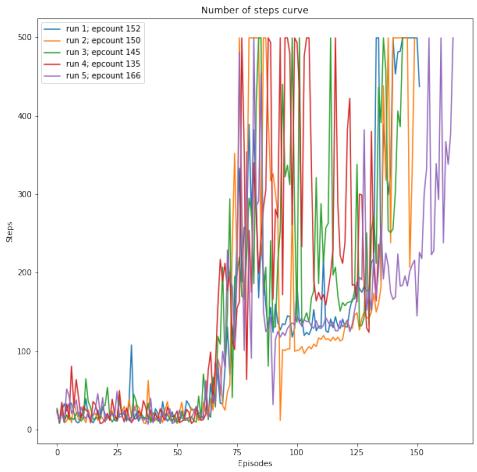
Experiment 6

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* increased to 256
- Now there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

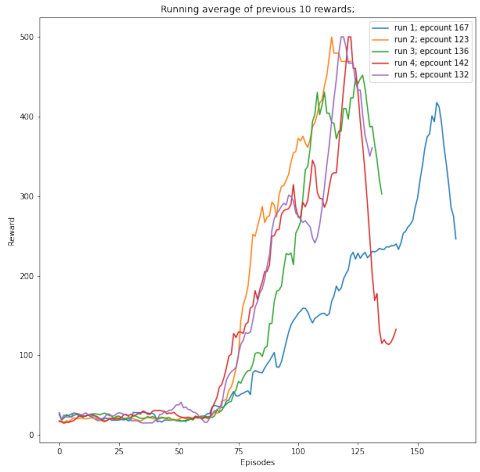
- Increase in batch size should have given better performance even though it took more time to converge
- But there is a dip in the performance(increase in the episode count) in this experiment, after increasing batch size further to 256

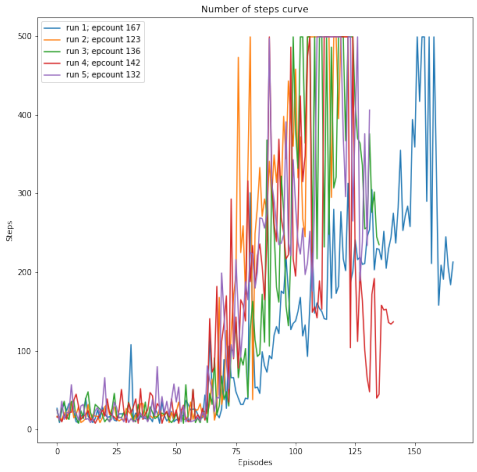
Experiment 7

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* is now fixed to 64 on comparing the performances from Experiments 4,5 and 6
- *Update every* is now varied to a value of 20
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

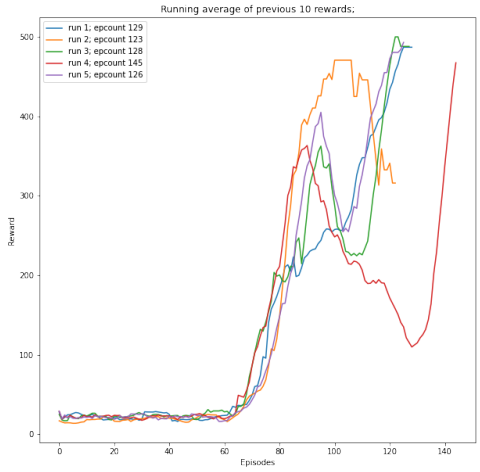
- Performance reduced a bit after changing the update frequency down to 20
- Reducing the update frequency might have slowed down the convergence rate, but the performance could have actually increased if other hyperparameters were tuned accordingly
- For this set of hyperparameters decided, reducing update frequency does not help in increasing performance

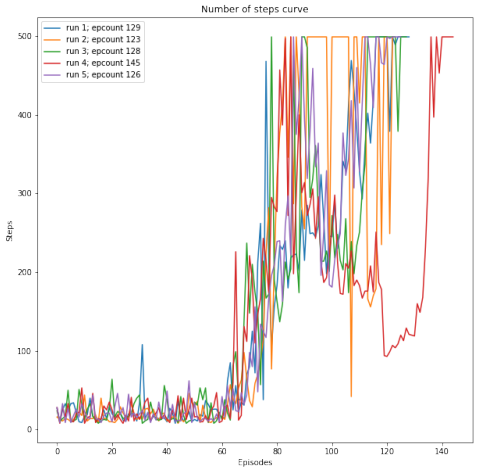
Experiment 8

Current Hyperparameters

- *Buffer size* is 10000 (Different from starting parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* is now varied to a value of 50
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

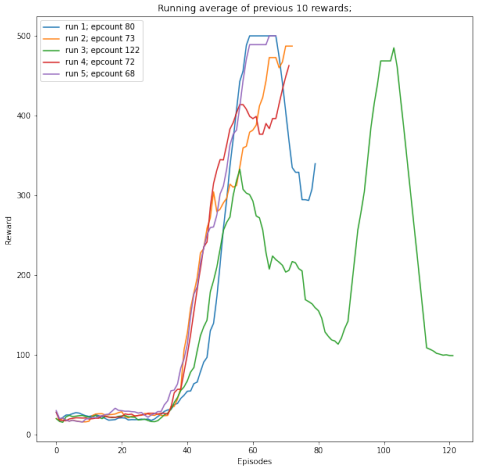
- For an update frequency of 50, performance is comparatively better than when the update frequency was 20.
- And also better than the performance at the update frequency of 100
- This shows that the update frequency of 50 is the best for the given set of other hyperparameters

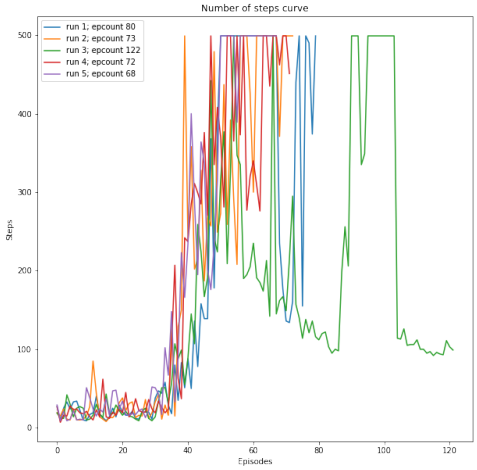
Experiment 9

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* will be changed to 50, as experiment 7 had increased performance
- *Decay factor* is now varied to a value of 0.9
- Now there are four changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

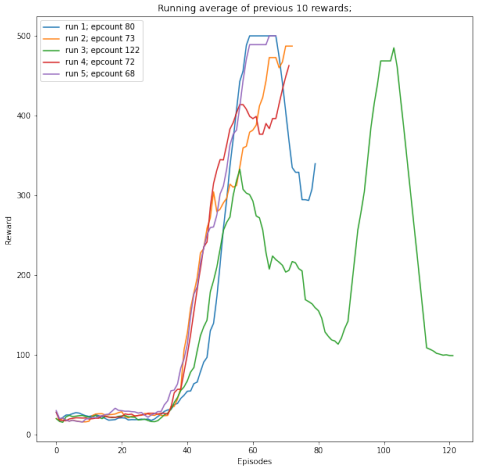
- Performance increased significantly on decreasing decay factor from 0.95 to 0.9
- This shows that the agent needed more exploration time to figure out the optimal policy, which it was not able to do till now with a 0.8 decay factor, and it was performing sub-optimal actions

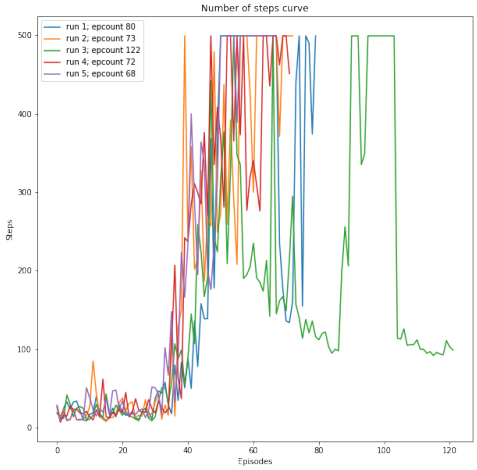
Experiment 10

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* is 50 (Different from starting parameter)
- *Decay factor* of value 0.9 improved the performance of the model
- *Epsilon end* is varied to $1 * 10^{-5}$
- Now there are five changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

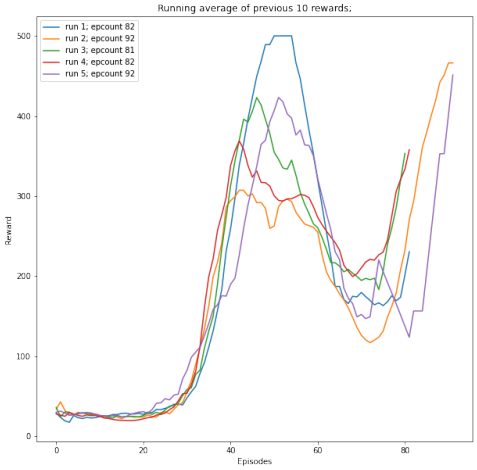
- There was an even better improvement in the agent learning after decreasing the epsilon end value to $1 * 10^{-5}$
- This shows that after a considerable amount of exploration,decreased epsilon end value helped the policy to exploit the best possible actions

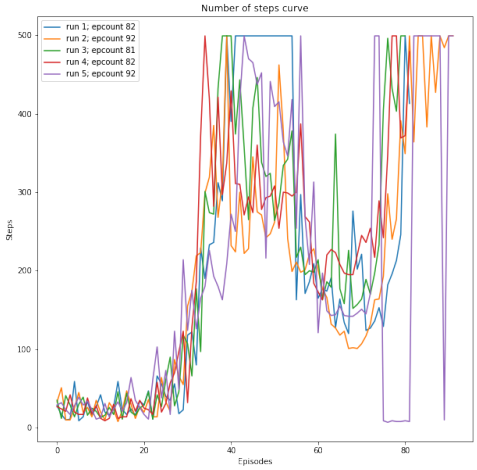
Experiment 11

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Update every* is 50 (Different from starting parameter)
- *Decay factor* is 0.9 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are seven changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

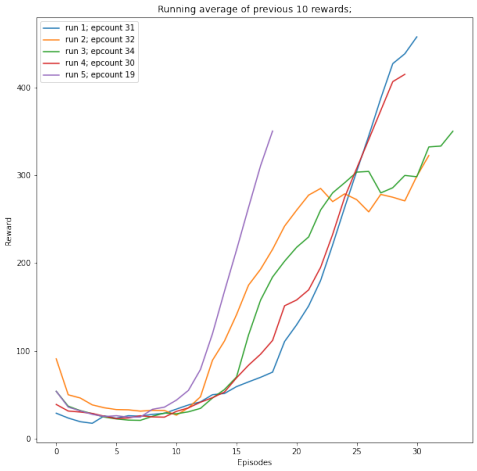
- Softmax exploration policy produced a very good result, even though computationally it took more time
- This change in policy, helped the agent to have better exploration for learning the optimal policy

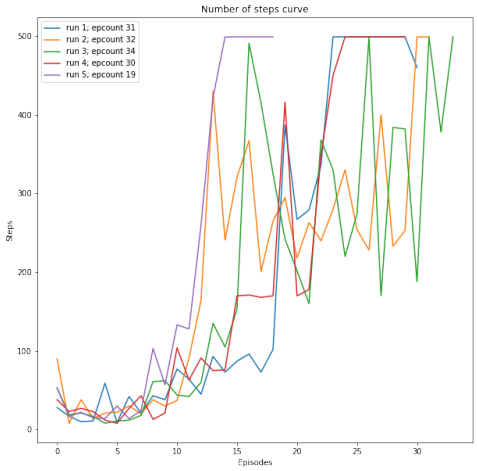
Experiment 12

Current Hyperparameters

- *Buffer size* is 10000 (Different from default parameter)
- *Batch size* is 128 (Different from starting parameter)
- *Decay factor* is 0.6 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are five changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

- Performance is much better than experiment 11 due to the low decay factor this time.
- This experiment outperformed epsilon greedy's best configuration(Experiment 10)

Best configuration for CartPole-v1 environment

Out of all the experiments, the 12th experiment had the best performance of 29.2, whereas starting parameters had a performance of 153.6

MountainCar-v0

General Technique

For choosing the below-starting hyperparameters, a few initial experiments were done. And it was found that:

- Altering network architecture did not have a significant effect, and there was also a trade-off in the time it took for each run with increasing layers and its sizes.
- Altering truncation limit did not have any effect on the performance, tested values as low as $1 * 10^{-4}$

Note: Instead of 10 runs, we have implemented five runs, as the experiments took a long time to run. And the results had good variations with just those five runs.

Reward shaping The default reward function was changed for this environment for better and faster reward convergence. The custom reward function used is as follows:

```
def reward_shaping(state, next_state):
sreward = 0
if (next_state[0] >= 0.5):
sreward += 400
sreward += 400*((np.sin(3*next_state[0]) * 0.0025 + 0.5 * next_state[1] *
                                        next_state[1]) - (np.sin(3*state[0]) * 0.
                                        0025 + 0.5 * state[1] * state[1]))
return sreward
```

Simply giving a positive reward once the car reaches the destination by trial and error in some episode and a negative reward for all other time steps will not work. It will take a pretty long time before the network learns the optimal strategy.

To reach the peak from the valley, the car needs to gain mechanical energy so the optimal strategy would be one in which the car gains mechanical energy (Potential energy + Kinetic energy) at every time step. So a good reward function would be the increase in mechanical energy at every time step.

Starting Hyperparameters

- hidden layers = 3
- hidden sizes $[64, 64, 128]$
- truncation limit = $1 * 10^{32}$
- learning rate(α) = $5 * 10^{-4}$
- discount factor (γ) = 0.99
- buffer size = 10000
- batch size = 64
- update every = 20
- exploration policy = epsilon-greedy

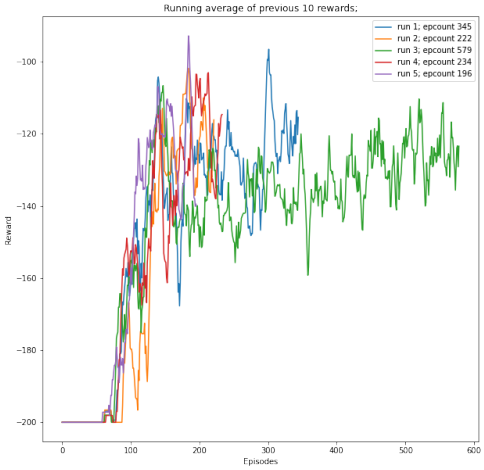
- decay factor $= 0.95$
- epsilon start $= 10$
- epsilon end $= 5 * 10^{-4}$

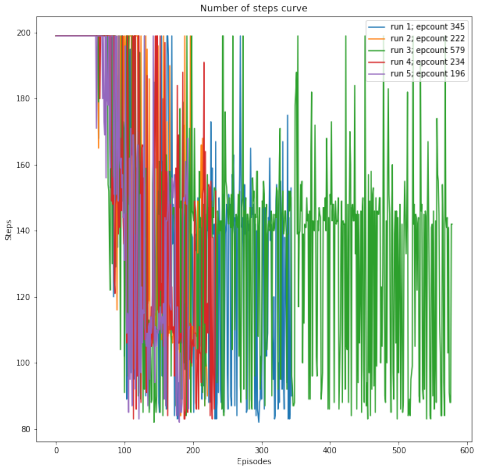
Experiment 1

Current Hyperparameters

- Here, we use the starting hyperparameters to perform the experiment.

Reward Curve and Steps Curve for the 5 runs





Justification

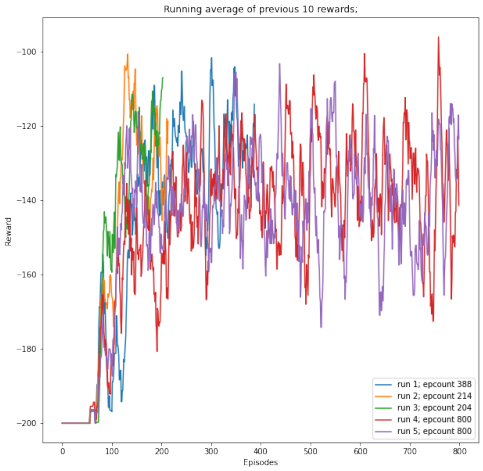
- From the reward and steps curve, it is evidently visible that the agent is almost reaching the threshold reward for all the runs for our choice of starting hyperparameters
- We came up with these default parameters after tweaking and experimenting around for some iterations.

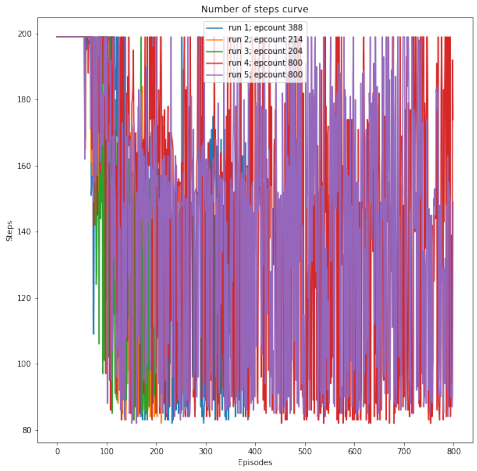
Experiment 2

Current Hyperparameters

- *Discount factor*(γ) is changed from 0.99 to 0.97 and tested for this experiment
- This is the only change from the starting hyperparameters.

Reward Curve and Steps Curve for the 5 runs





Justification

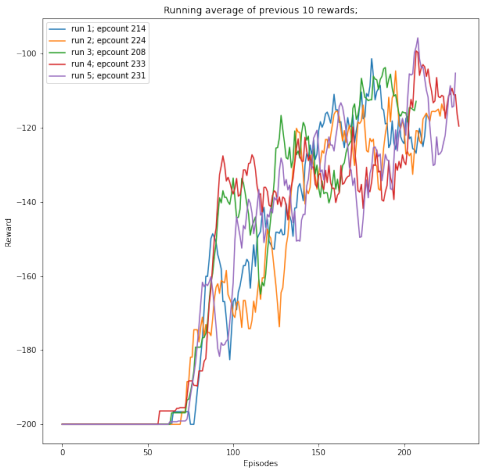
- In this experiment, there is an overall increase in the number of episodes before reaching the threshold reward with the maximum episode count being 800(implying the run itself got over) in the fourth and fifth run
- This is due to the change in discount factor by 0.02, from 0.99 to 0.97. This is because a lesser discount factor led the agent to focus on immediate rewards than the long-term rewards

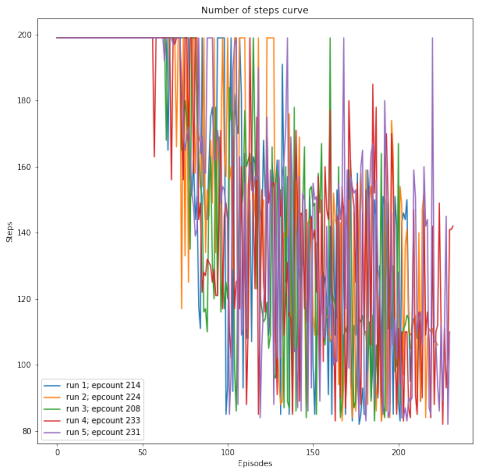
Experiment 3

Current Hyperparameters

- *Discount factor* resorted back to the original value(0.99), since experiment 2 did not show improved performance
- *Learning rate* changed to $9 * 10^{-4}$
- There is only one change from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

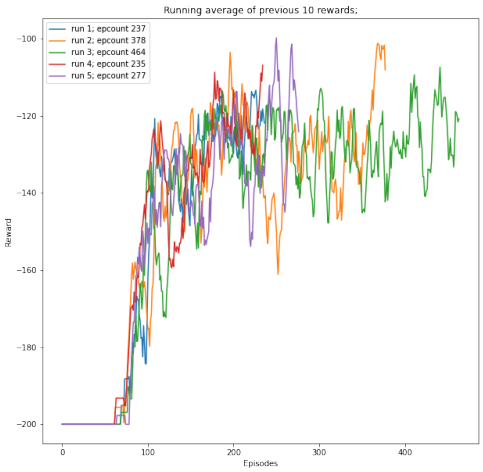
- There is a slight decrease in the episode count in this experiment than Experiment 1
- Thus it shows that, increasing the learning rate did help the agent to converge faster, without any overshooting occuring

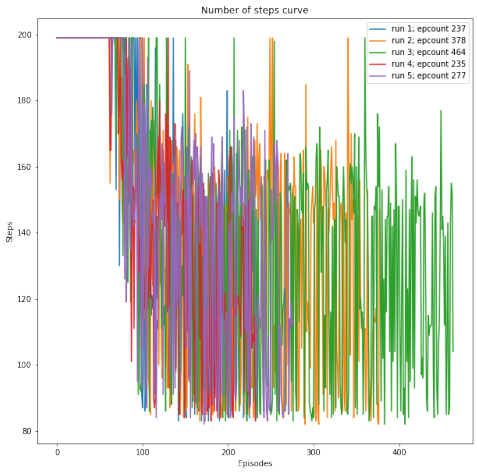
Experiment 4

Current Hyperparameters

- *Learning rate* of $9 * 10^{-4}$ improved the performance
- *Buffer size* increased to 40000
- There are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

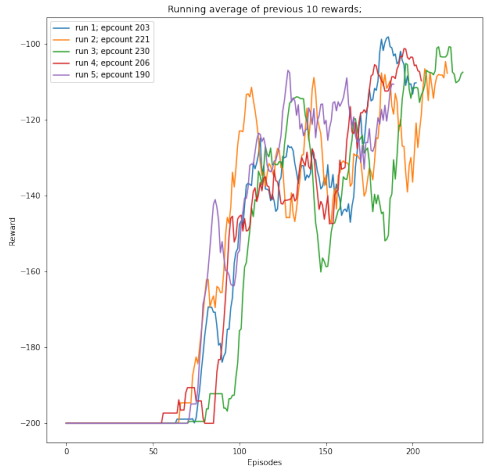
- There is a decrease in the performance(increase in the episode count) in this experiment after increasing the buffer size to 40000
- Higher buffer size contains more experiences in the past, and this decline in performance shows that for the Mountaincar environment, recent information is more relevant

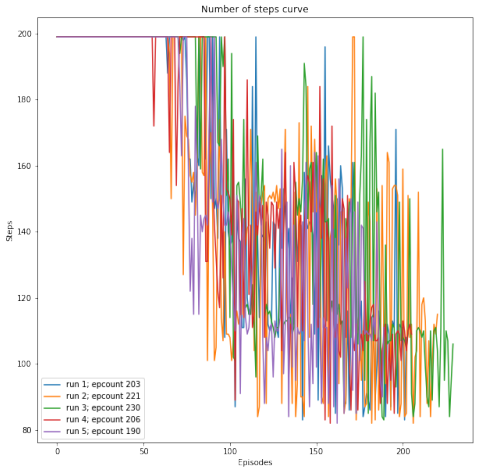
Experiment 5

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Buffer size* resorted back to original value(10000),since 40000 did not improve the performance.
- *Batch size* increased to 128
- Now, there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

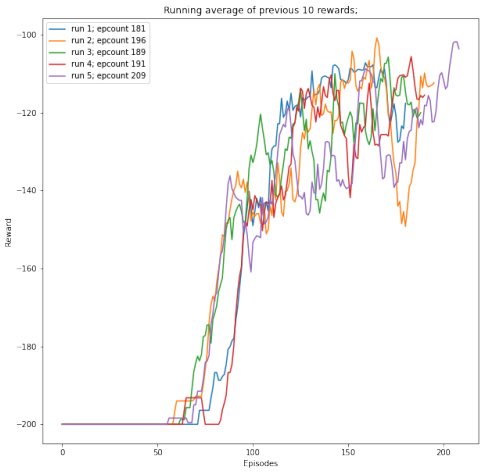
- There is a considerable increase in the performance(decrease in the episode count) in this experiment,after increasing batch size to 128
- Larger batch size has helped reduce the variance in gradient estimates leading to better performance

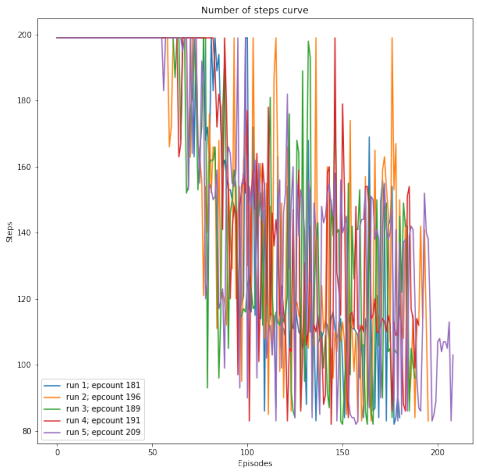
Experiment 6

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* increased to 256
- Now there are two changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

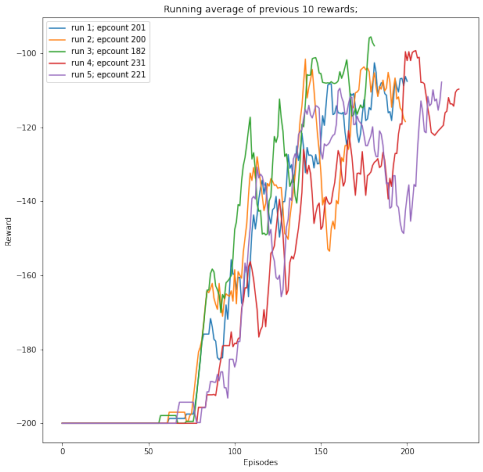
- Further increase in the batch size from 128 to 256 increased the performance even though it took computationally more time to produce this result
- Larger batch size has helped reduce the variance in gradient estimates leading to better performance

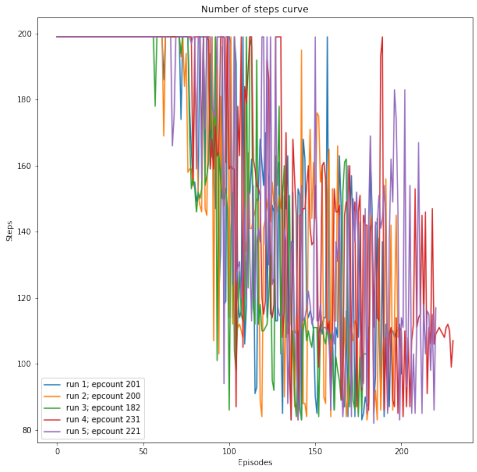
Experiment 7

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is now fixed to 256 on comparing the performances from Experiments 4,5 and 6
- *Update every* is now varied to a value of 50
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

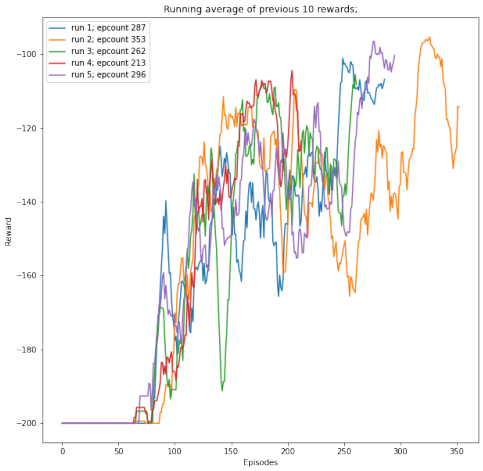
- Performance reduced a bit after raising the update frequency to 50
- May be an increase in update frequency might have led to instability in the training process, as the network may not have had enough time to properly learn from the experiences in the replay buffer before the weights are updated

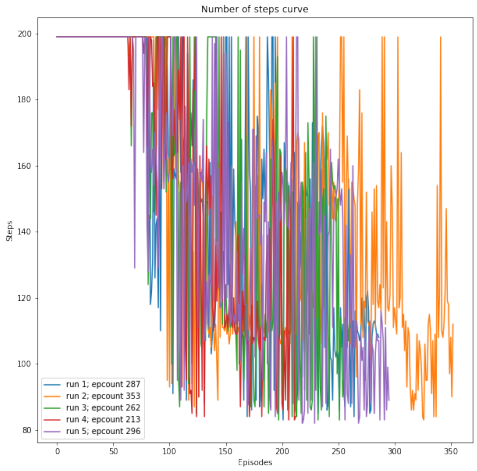
Experiment 8

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is 256 (Different from starting parameter)
- *Update every* is now varied to a value of 100
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

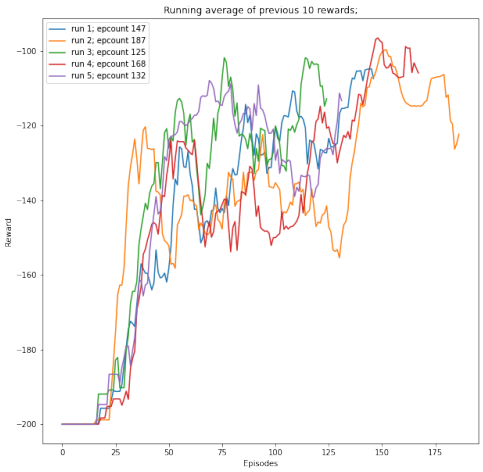
- Performance reduced even more after raising the update frequency to 100
- This is not surprising since there was already a dip in performance for an update frequency of 50,and even more dip for a value of 100 proves the actual working of update frequency

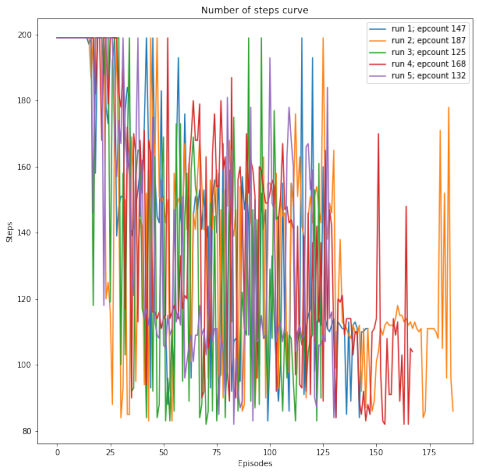
Experiment 9

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is 256 (Different from starting parameter)
- *Update every* will remain 20(Starting parameter), as varying it in Experiments 7 and 8 did not improve the performance
- *Decay factor* is now varied to a value of 0.8
- Now there are three changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

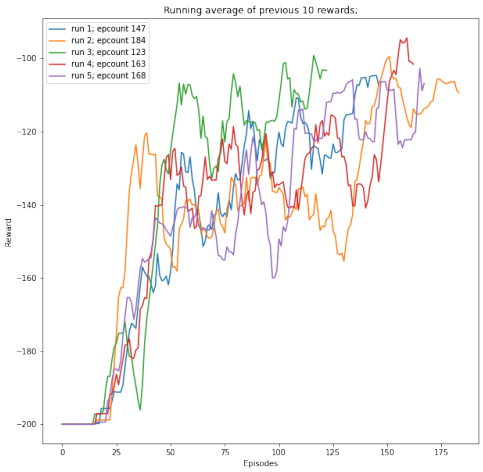
- Performance increased significantly on decreasing decay factor from 0.95 to 0.8
- This shows that the agent needed more exploration time to figure out the optimal policy, which it was not able to do till now with a 0.95 decay factor, and it was performing sub-optimal actions

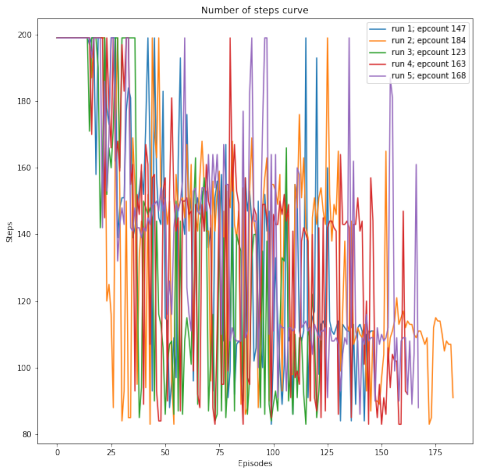
Experiment 10

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is 256 (Different from starting parameter)
- *Decay factor* of value 0.68improved the performance of the model
- *Epsilon end* is varied to $1 * 10^{-5}$
- Now there are four changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

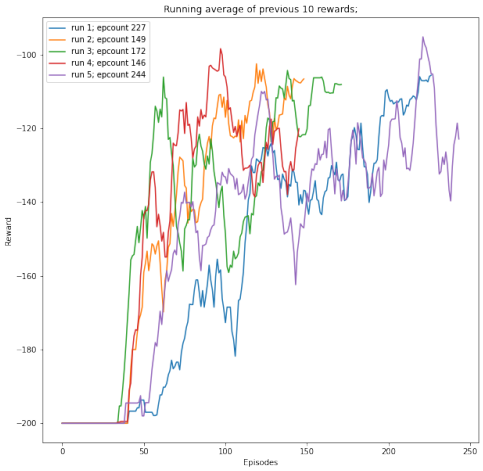
- There was an even better improvement in the agent learning after decreasing the epsilon end value to $1 * 10^{-5}$
- This shows that after a considerable amount of exploration, decreased epsilon end value helped the policy to exploit the best possible actions

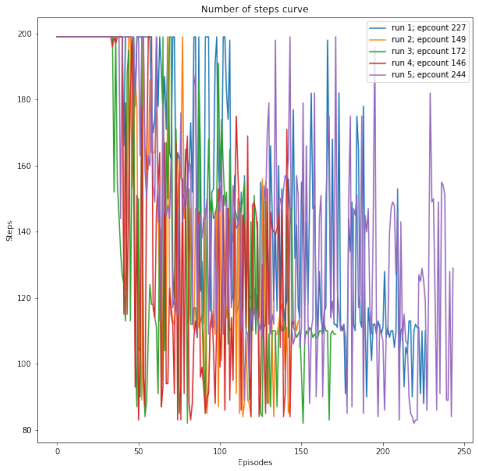
Experiment 11

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is 256 (Different from starting parameter)
- *Decay factor* is 0.8 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are six changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

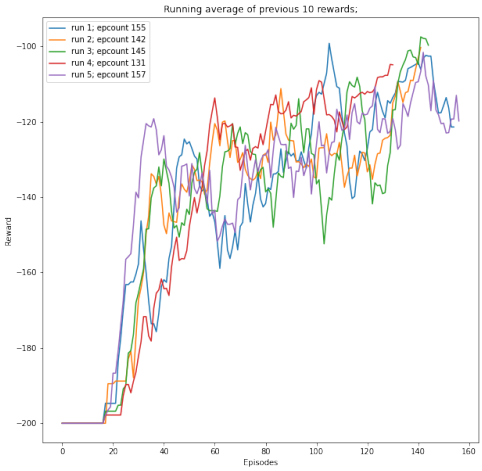
- Softmax exploration policy produced a very good result, even though computationally it took more time
- This change in policy, helped the agent to have better exploration for learning the optimal policy

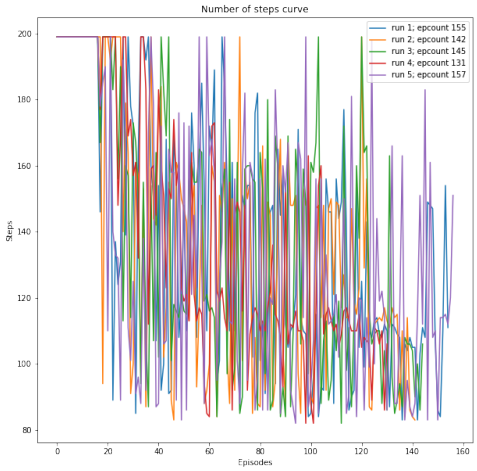
Experiment 12

Current Hyperparameters

- *Learning rate* is $9 * 10^{-4}$ (Different from starting parameter)
- *Batch size* is 256 (Different from starting parameter)
- *Decay factor* is 0.6 (Different from starting parameter)
- *Epsilon end* of value $1 * 10^{-5}$ improved the performance
- *Epsilon start* is varied to 40
- Exploration policy is changed to *softmax*
- Now there are six changes from the starting hyperparameters

Reward Curve and Steps Curve for the 5 runs





Justification

- Performance is much better than experiment 11 due to the low decay factor this time.
- This experiment outperformed epsilon greedy's best configuration(Experiment 10)

Best configuration for MountainCar environment

Out of all the experiments, the 12th experiment had the best performance of 146.0, whereas starting parameters had a performance of 315.2

Conclusion for DQN

- Based on the conducted experiments, it can be concluded that the DQN technique is most effective for the Acrobot environment and least effective for the MountainCar environment.
- **Decay factor** is the most performance influencing hyperparameter of all
- **Reward shaping** helped the agents to explore the potentially rewarding states and also helped in mitigating the sparse rewards that the default reward function would provide
- For all three environments, a smaller **buffer size** was found to be more preferable, indicating that the agent relied more heavily on recent information rather than past experiences.
- Although we did not extensively experiment with **network architecture** for DQN due to the time-consuming nature of the training process, it is important to adjust these hyperparameters as they can significantly impact the performance of the system.

From the experiments we conducted, we found that DQN is a better Deep RL technique for these environments than Actor-Critic methods which will be evident from the results shown in the following section

Actor-Critic (AC) Methods

Description and Code snippets

- As reward shaping did not improve the results, we do no reward shaping for any environment in the AC experiments.
- The 1-step code was borrowed from tutorial 5 and it was modified for the 2 new variations.
- For the n-step AC and full return AC, we used the Huber Loss function for the critic loss rather than MSE Loss. As discussed by the professor, this helped in reducing the impact of outliers during optimization.
- The code snippet for n-step returns AC is shown below. Clearly, the below code takes in the rewards per step list, the values list (which has $V(s_t)$ for all steps t) and computes the array of n-step returns. Here, $n = STEP$ is a global variable.

```
def compute_nstep_returns(self, rewards, values, gamma = 0.99):
"""Compute n step returns at all timesteps."""
  n = tf.shape(rewards)[0]
  returns = tf.TensorArray(dtype=tf.float32, size=n)
  rewards = tf.cast(rewards[::-1], dtype=tf.float32)
  disc_return = tf.constant(0.0)
  disc_ret_shape = disc_return.shape

  # loop to compute the return for every instant
  for i in tf.range(n):
    if i+STEP >= n:
      ret = tf.constant(0.0)
      ret_shape = ret.shape
      for j in tf.range(i,n):
        ret = tf.math.pow(gamma,float(j-i))*rewards[j] + ret
        ret.set_shape(ret_shape)
      disc_return = ret
```

```
      else:
        ret = tf.constant(0.0)
        ret_shape = ret.shape
        for j in tf.range(i,i+STEP):
          ret = tf.math.pow(gamma,float(j-i)) * rewards[j] + ret
          ret.set_shape(ret_shape)
        disc_return = ret + tf.math.pow(gamma,float(STEP)) * values[i+STEP]
      disc_return.set_shape(disc_ret_shape)
      returns = returns.write(i, disc_return)
    returns = returns.stack()
    return returns
```

- The code snippet for full returns AC is shown below. Again, the below code takes in the rewards per step list, the values list (which has $V(s_t)$ for all steps t) and computes the array of full returns.

```
def compute_full_return(self, rewards, gamma = 0.99):
"""Compute full returns at all timestep."""
  n = tf.shape(rewards)[0]
  returns = tf.TensorArray(dtype=tf.float32, size=n)

  # loop to calculate full returns for all time steps
  rewards = tf.cast(rewards[::-1], dtype=tf.float32)
  disc_ret = tf.constant(0.0)
  disc_ret_shape = disc_ret.shape
  for i in tf.range(n):
    reward = rewards[i]
    disc_ret = reward + gamma * disc_ret
    disc_ret.set_shape(disc_ret_shape)
    returns = returns.write(i, disc_ret)
  returns = returns.stack()[::-1] # reverse return array for correct order
                                  .
  return returns
```

- After the returns are computed, the TD error (δ) is computed by subtracting `values` from the `returns`. The actor loss and critic loss are calculated using direct `Tensorflow` functions.
- Also, for the 2 new variations, firstly, an episode of experience is collected using the actor's policy and then, the updates for all timesteps of the episode are made in a batch.

Note: Variance plotted for the below performed experiments are in fact the standard deviations of the rewards

Acrobot-v1

General Technique

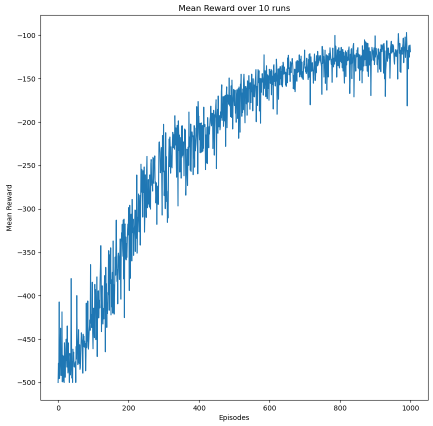
- As we are restricted to show only the top 12 experiments in our report, we will be showing 3 experiments each for the 1-step, n-step for $n = 5, 10$ and full return variations.
- The experiments are around changing the network architecture(number of hidden layers and their sizes) and the learning rate α.

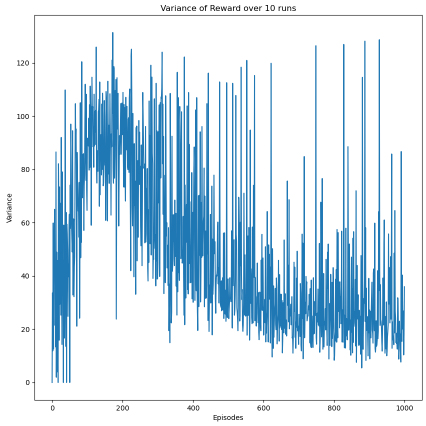
1 Step AC Experiment 1

Hyperparameters

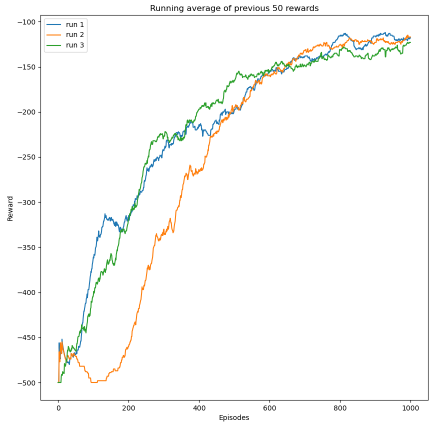
- hidden layer count $= 1$
- hidden layer sizes $= [1024]$
- learning rate $\alpha = 1 * 10^{-4}$

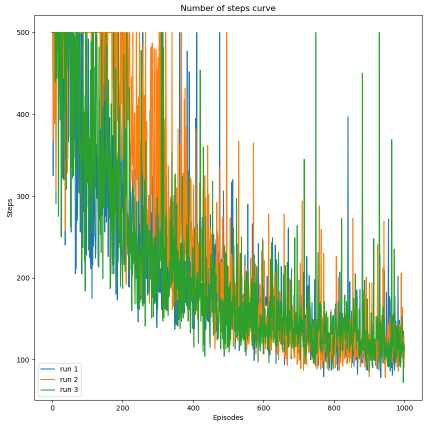
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

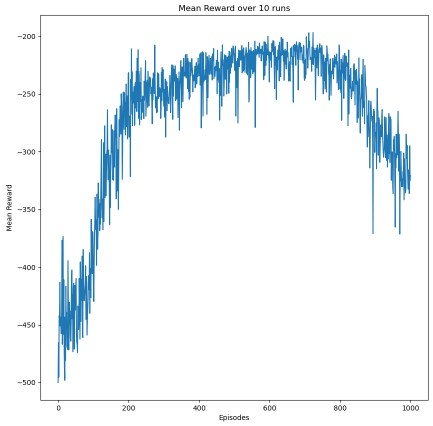
- Mean reward is almost asymptotic and variance has also reduced but with oscillations still present
- This set of hyperparameters seem to produce maximum achievable optimality in the policy

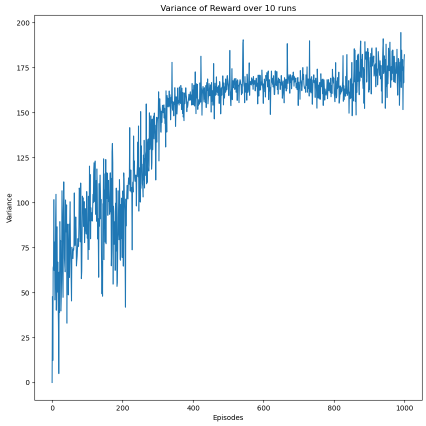
1-Step AC Experiment 2

Hyperparameters

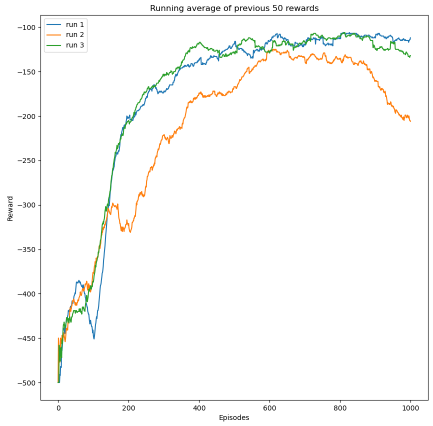
- hidden layer count $= 1$
- hidden layer sizes $= [1024]$
- learning rate $\alpha = 2 * 10^{-4}$

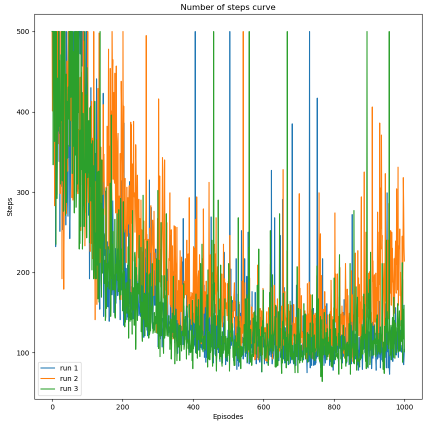
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

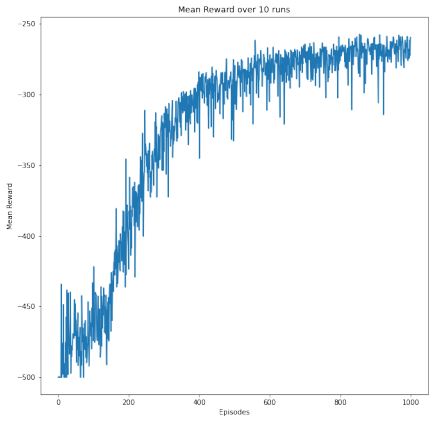
- Increase in learning rate made the reward curve to dip again after reaching an average reward of -200
- Variance also has increase considerably than the previous experiment
- Thus it shows that increasing the learning rate did not help the agent to learn, but rather made it bounce around the optimal policy.

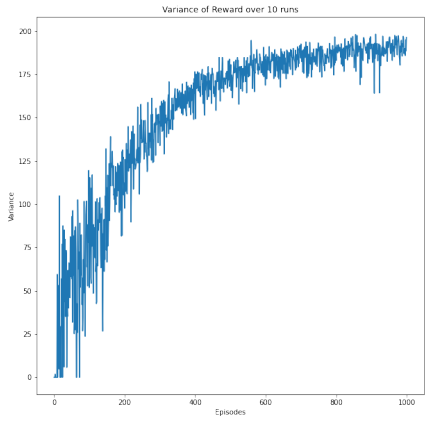
1-Step AC Experiment 3

Hyperparameters

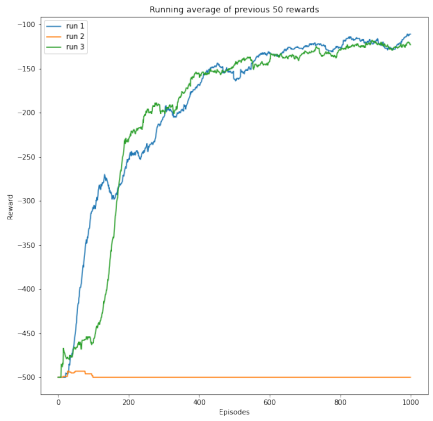
- hidden layer count = 1
- hidden layer sizes = [2048]
- learning rate $\alpha = 1 * 10^{-4}$

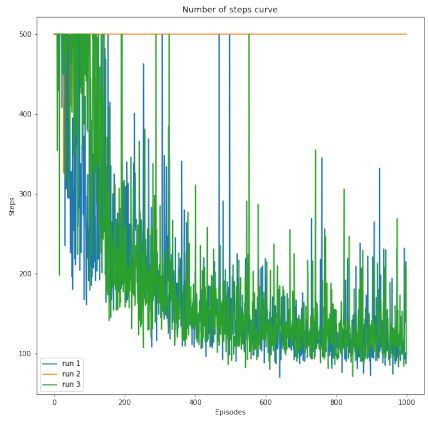
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

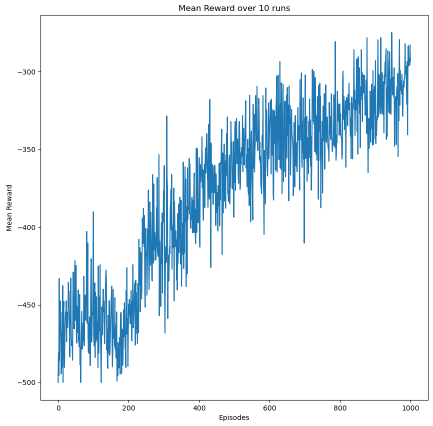
- Mean reward is almost asymptotic two of the runs,but one run did not learn at all, leading to a very high variance in the end
- This might be due to the fact that the critic is providing bad guidance/feedback to the actor network leading to instability in the learning process

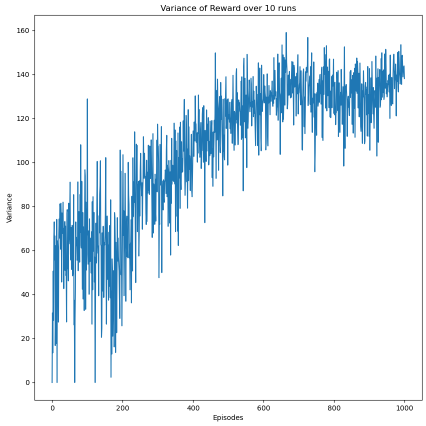
5-Step AC Experiment 1

Hyperparameters

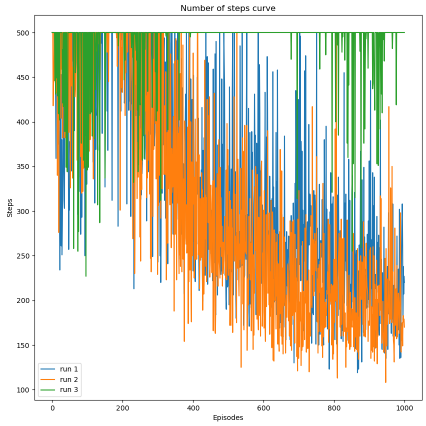
- hidden layer count $= 1$
- hidden layer sizes $= [1024]$
- learning rate $\alpha = 1 * 10^{-4}$

Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started



Justification

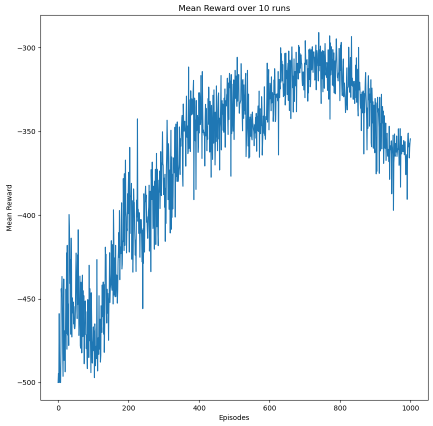
- Reward curve is in increasing trend for two of the runs,but one run did not learn at all, leading to a very high variance in the end
- This might be due to the fact that the critic is providing bad guidance/feedback to the actor network leading to instability in the learning process

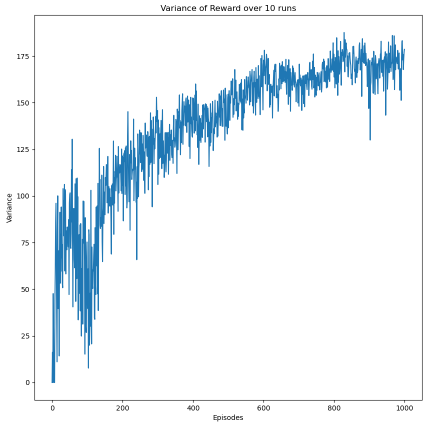
5-Step AC Experiment 2

Hyperparameters

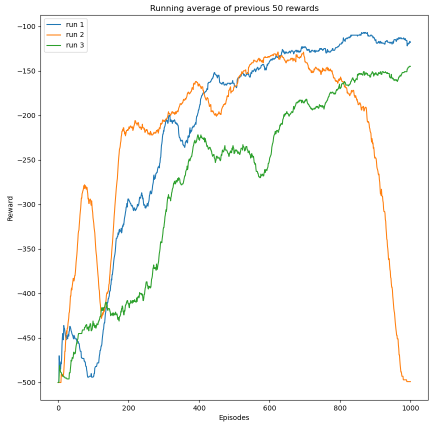
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

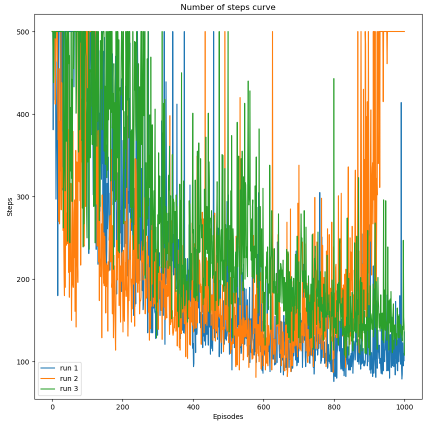
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

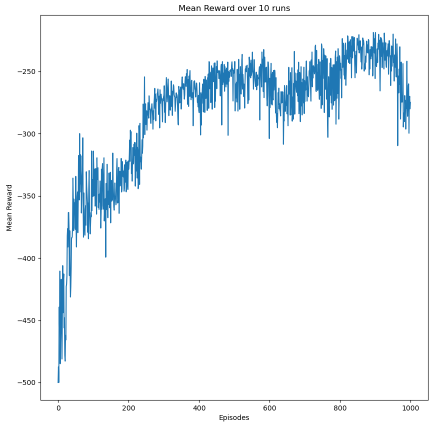
- Increasing the number of layers did not bring in much improvement in the performance
- Infact, now the variance is even higher than experiment 1

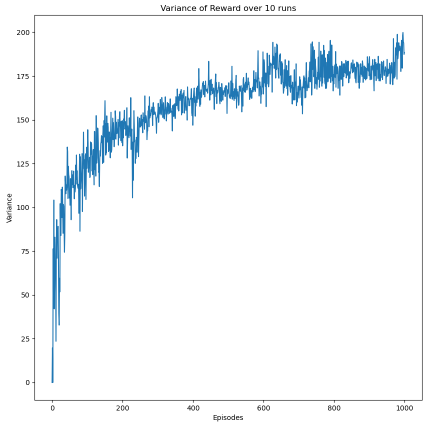
5-Step AC Experiment 3

Hyperparameters

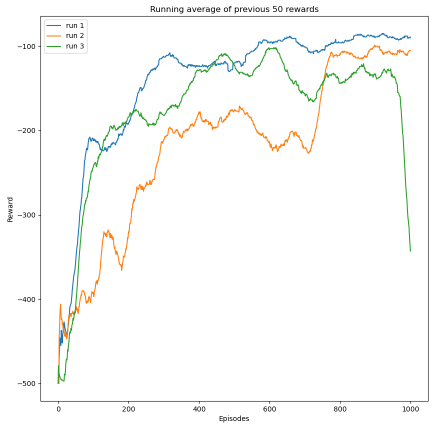
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 2 * 10^{-4}$

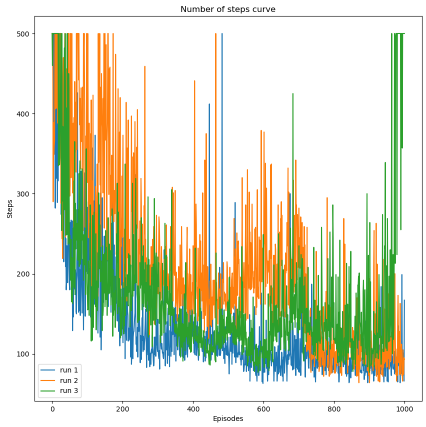
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

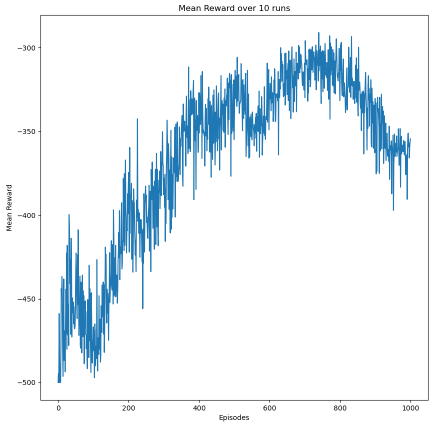
- Increasing the number of layers did not bring in much improvement in the performance
- Infact, now the variance is even higher than experiment 1 and better reward values
- Thus it shows that, increasing the learning rate did help the agent to converge faster, without any overshooting occuring
- There was a sudden dip in one of the runs in both experiment 2 and 3, which is an unexpected behaviour

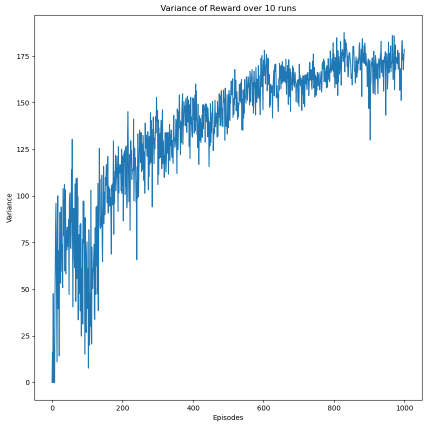
10-Step AC Experiment 1

Hyperparameters

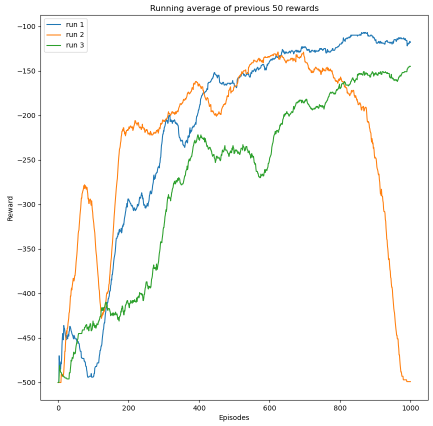
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

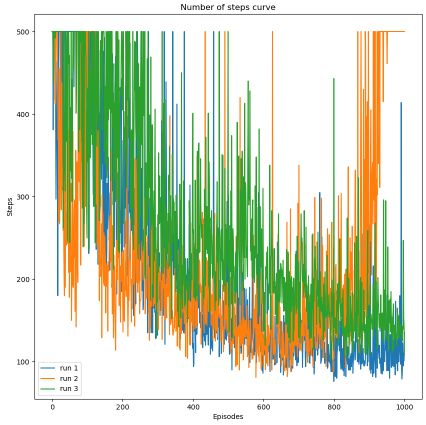
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

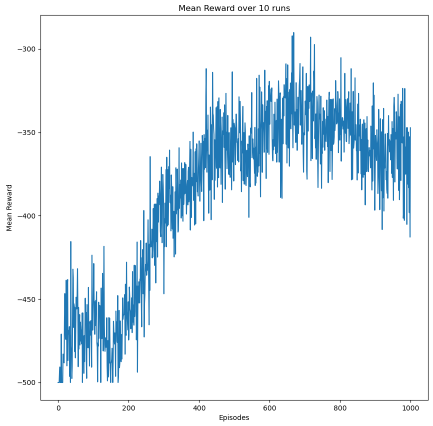
The behaviour of the agent is exactly similar to the behaviour in 5 step Actor-critic experiment for the same set of hyperparameters

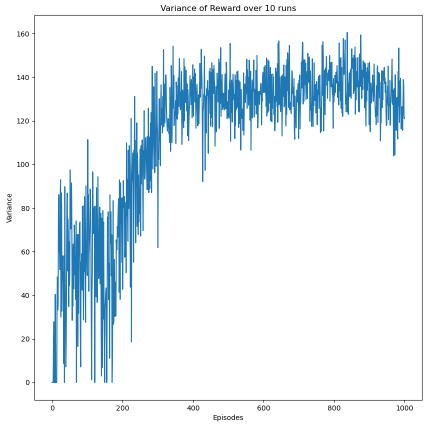
10-Step AC Experiment 2

Hyperparameters

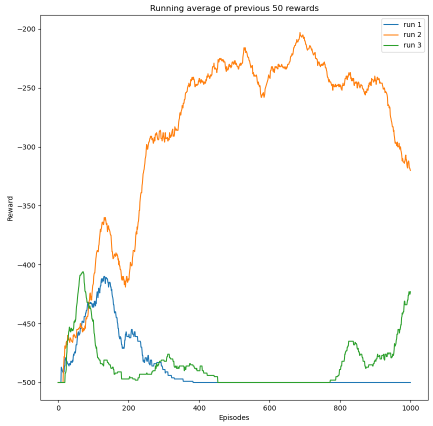
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 0.5 * 10^{-4}$

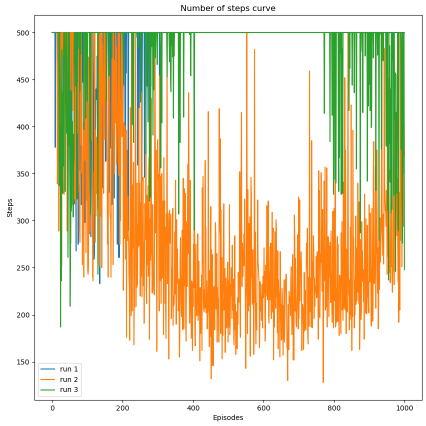
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

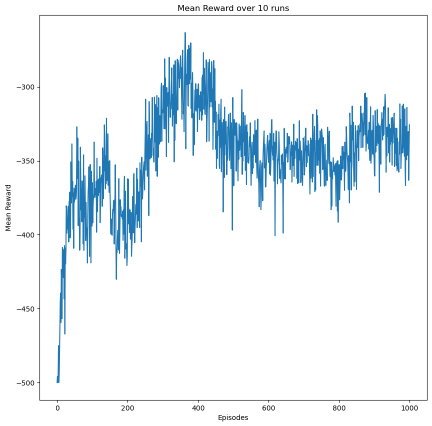
- Decreasing in learning rate has decreased the performance drastically.Out of the three runs, two did not perform well. One tried performing but behaviour is still suboptimal
- Lowering of Learning rate has led to low to no convergence at all.

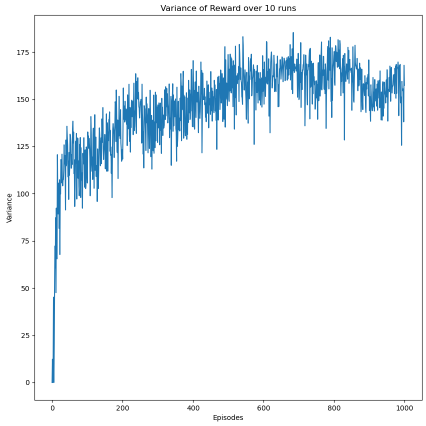
10-Step AC Experiment 3

Hyperparameters

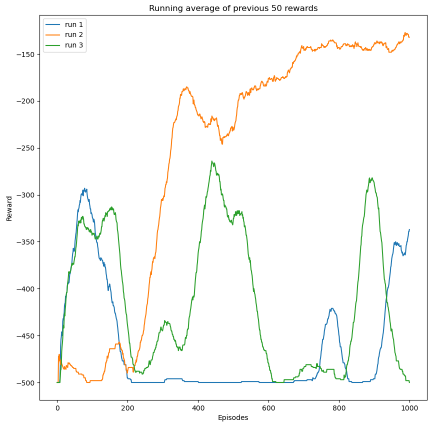
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 1024]$
- learning rate $\alpha = 0.5 * 10^{-4}$

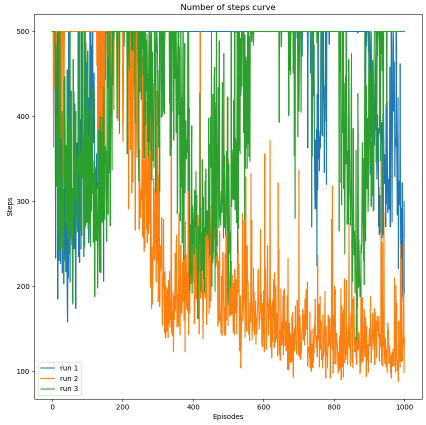
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

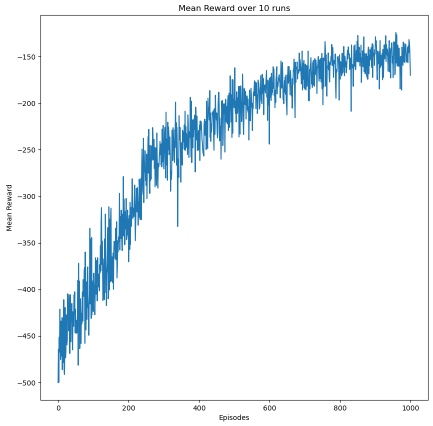
- Decreasing in learning rate has decreased the performance drastically.
- This is balanced by the increase in the layer sizes, but even that does not seem to help the performance dip caused by the learning rate.
- Two of the runs are caught in the loop of learning and failing

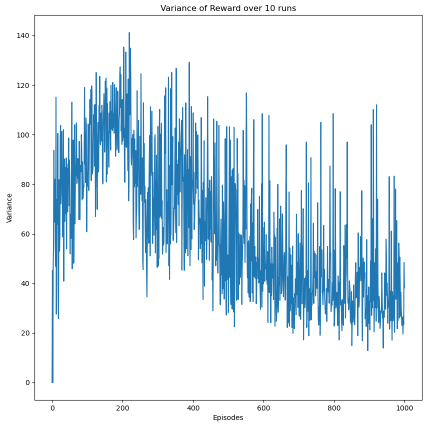
Full Return AC Experiment 1

Hyperparameters

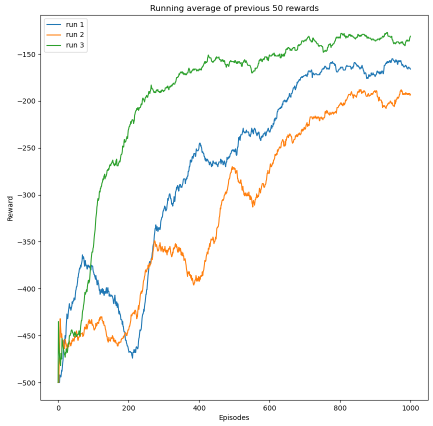
- hidden layer count = 1
- hidden layer sizes = [1024]
- learning rate $\alpha = 1 * 10^{-4}$

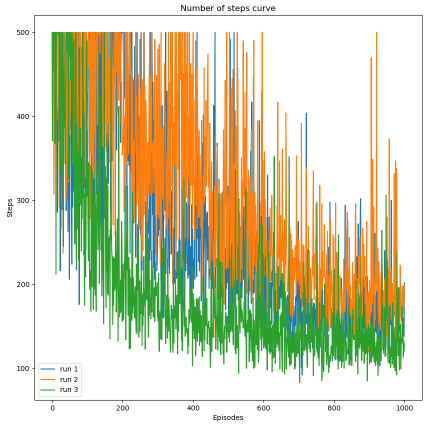
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

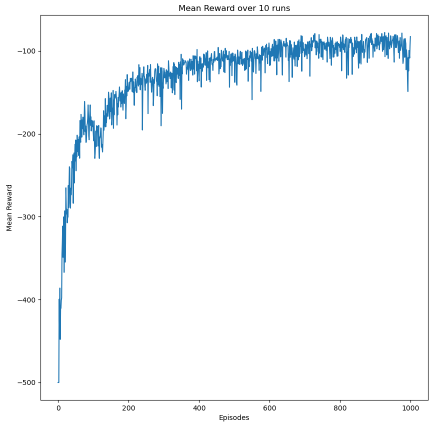
- Mean reward is increasing and variance has also reduced but with oscillations still present
- But the reward curves seem to be converging, given more number of episodes before termination, it might produce better rewards but performance will still be low

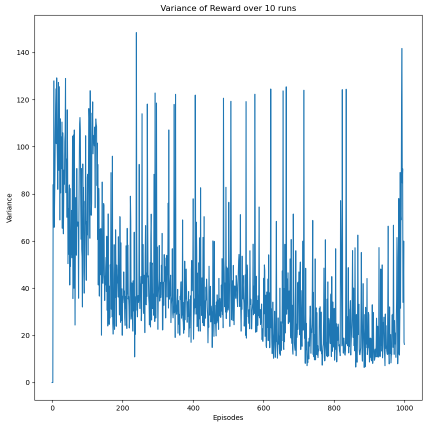
Full Return AC Experiment 2

Hyperparameters

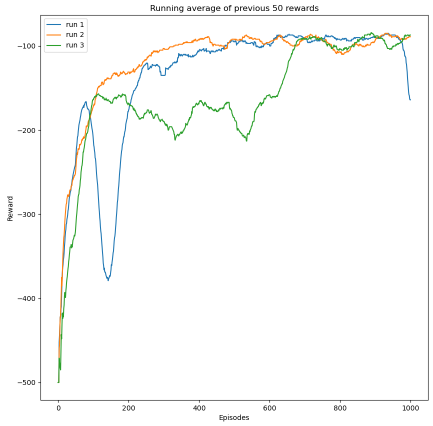
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 2 * 10^{-4}$

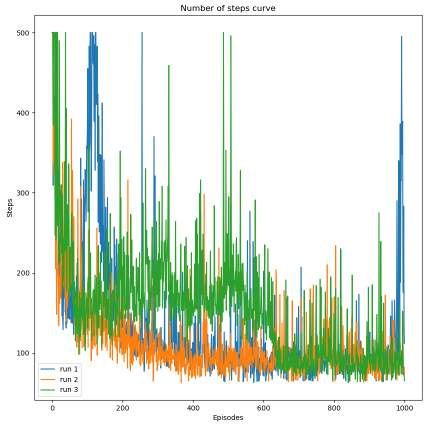
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

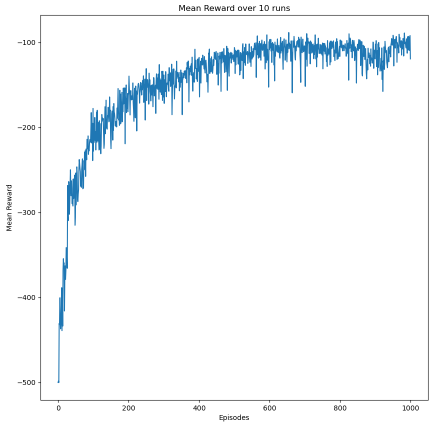
- Increasing the number of layers did bring in some improvement in the performance
- The reward curves have coincided leading to lower variance towards the end

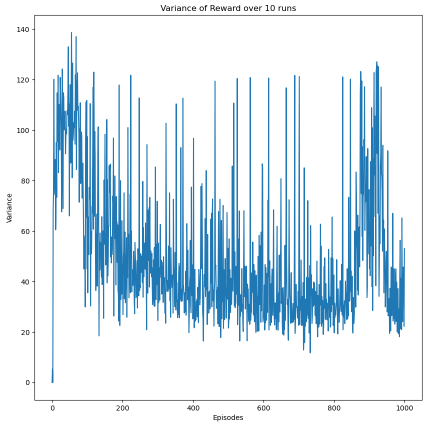
Full Return AC Experiment 3

Hyperparameters

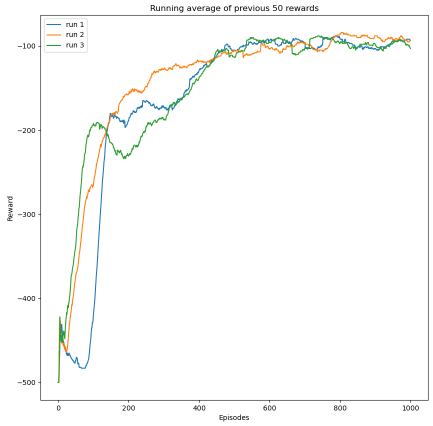
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 1024]$
- learning rate $\alpha = 1 * 10^{-4}$

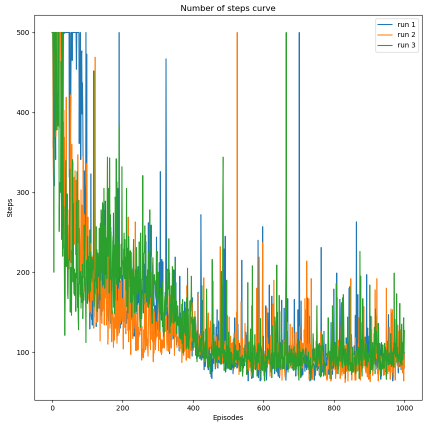
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

- Mean reward is almost asymptotic and variance has also reduced but with oscillations still present
- This set of hyperparameters seem to produce maximum achievable optimality in the policy

Overall Comments for Acrobot environment

- 1-step, 5-step and 10-step results were all similar with respect to number of steps in achieving the upright position of Acrobot
- Variation in variance also occured similarly across the AC variations with changes in the architecture and learning rate
- Full return Actor critic method had the best outputs of all the four variations with increasing and stabilizing reward curves with lower variance but took computationally more time to run

CartPole-v1

General Technique

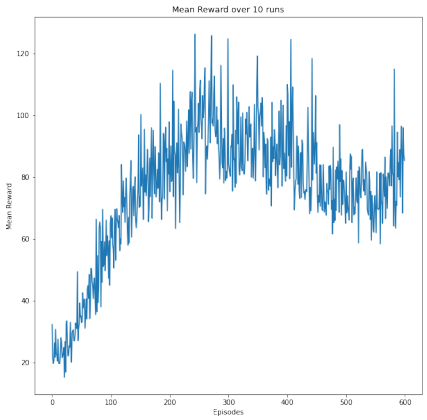
- As we are restricted to show only the top 12 experiments in our report, we will be showing 3 experiments each for the 1-step, n-step for $n = 5, 10$ and full return variations.
- The experiments are around changing the network architecture(number of hidden layers and their sizes) and the learning rate α.

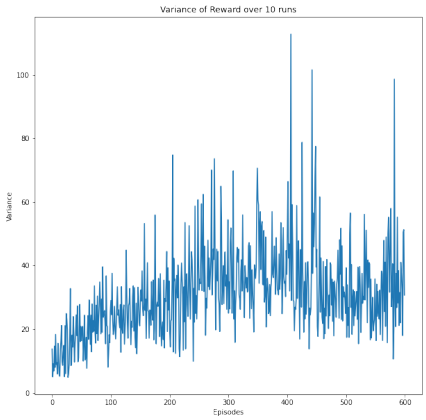
1 Step AC Experiment 1

Hyperparameters

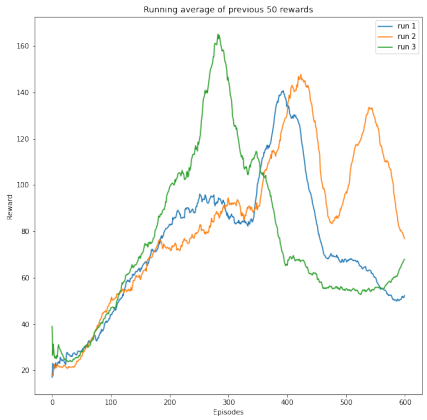
- hidden layer count = 1
- hidden layer sizes = [1024]
- learning rate $\alpha = 1 * 10^{-4}$

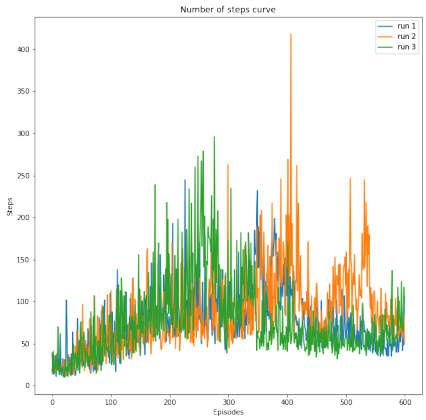
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

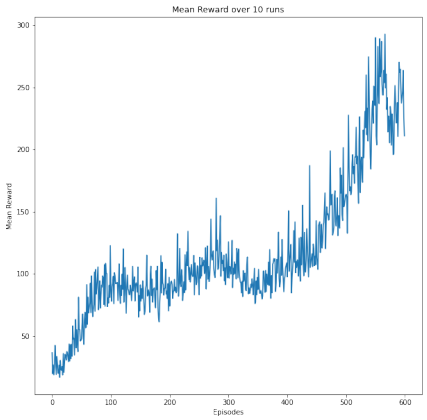
- For this set of hyperparameters, the reward curve reaches a maximum and starts declining,and variance also reduces
- This is not an optimal learning,but this one of the best performing compared various other experiments done

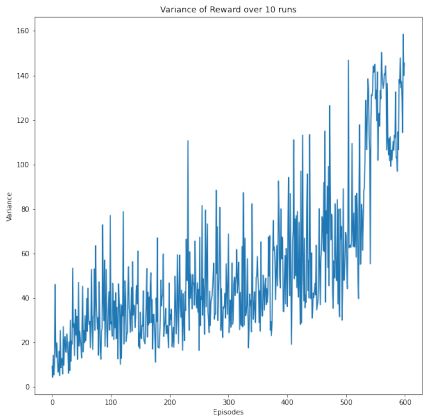
1-Step AC Experiment 2

Hyperparameters

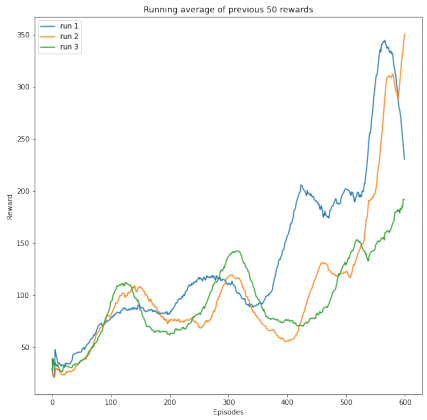
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 0.5 * 10^{-4}$

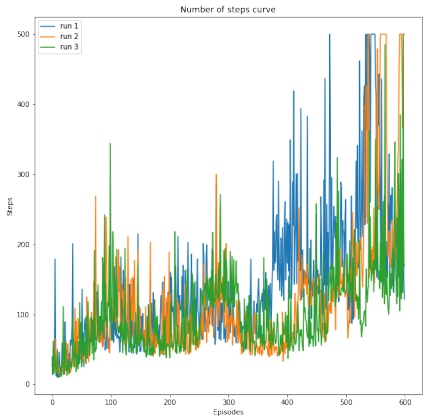
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

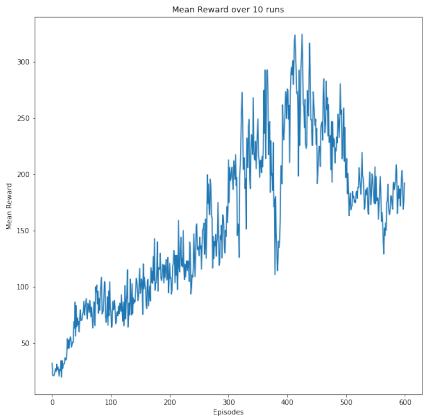
- Increase in the network size, seem to produce better reward
- This is not an optimal learning,but this one of the best performing set compared to various other experiments done

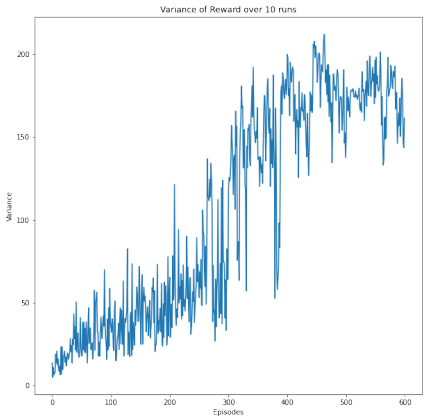
1-Step AC Experiment 3

Hyperparameters

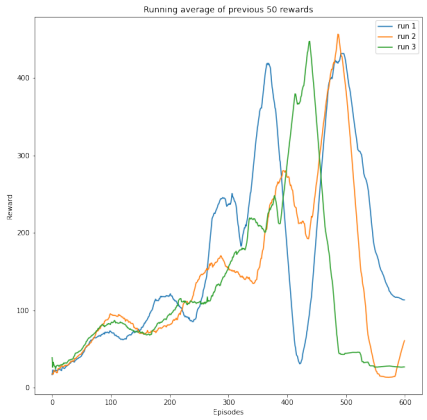
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 1024]$
- learning rate $\alpha = 0.5 * 10^{-4}$

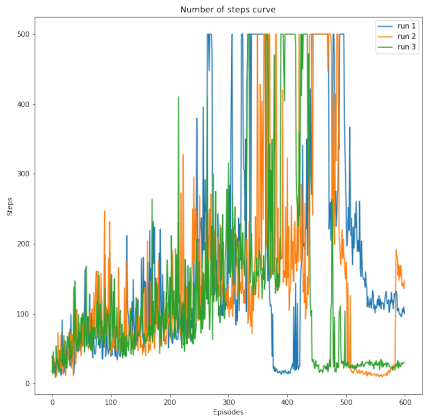
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

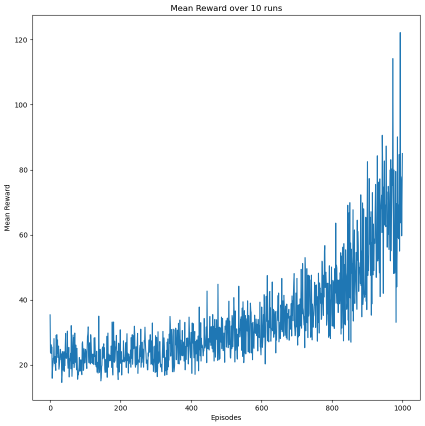
- Increase in the network size has led to increase in reward ,but it dropped again leading and oscillating very largely
- This large oscillations has also resulted in increased variance around the end of the run

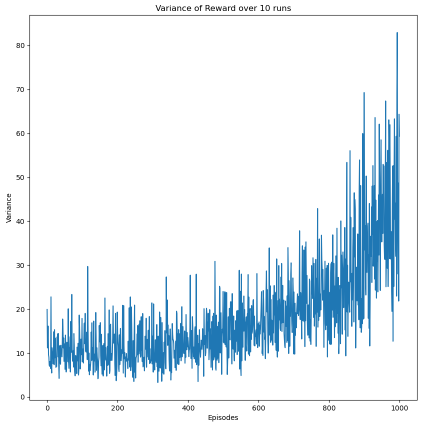
5-Step AC Experiment 1

Hyperparameters

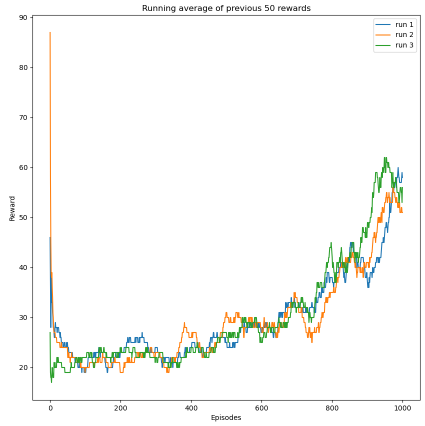
- hidden layer count = 1
- hidden layer sizes = [1024]
- learning rate $\alpha = 1 * 10^{-4}$

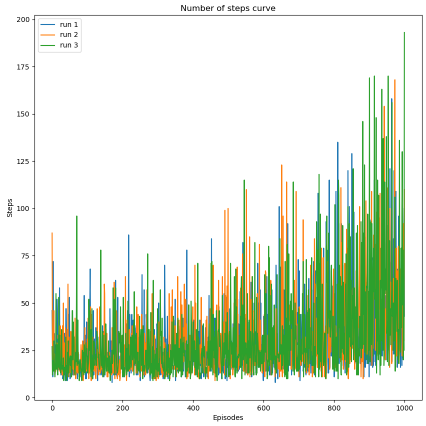
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

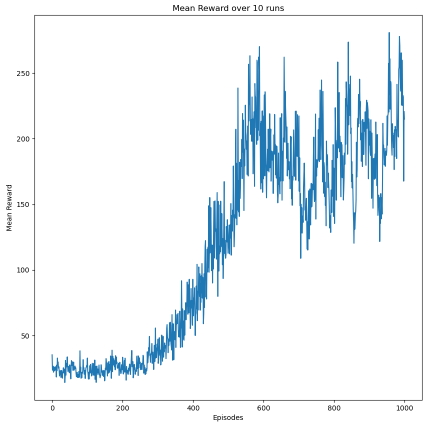
- Very slow convergence leading to a very slow increase in the reward curve for this network architecture
- Variance is also very low compared to one step actor critic experiment

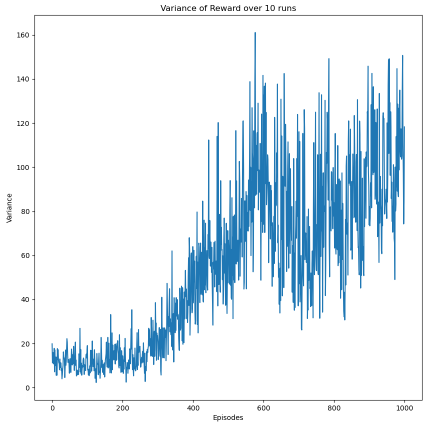
5-Step AC Experiment 2

Hyperparameters

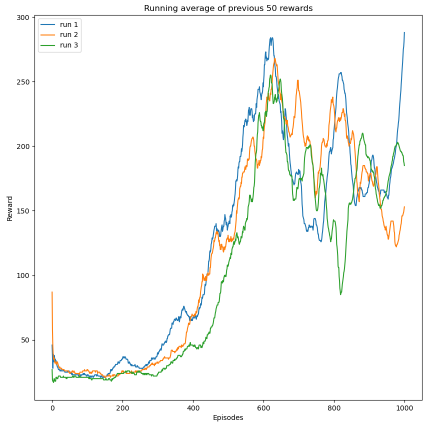
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

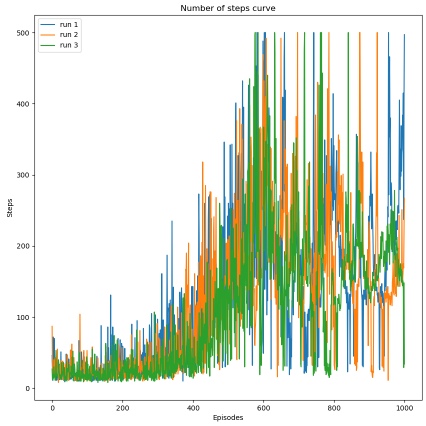
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

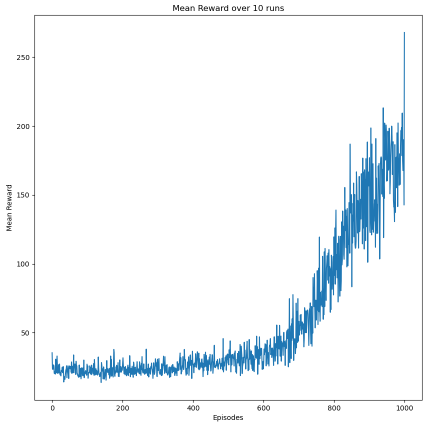
- Increase in the network size, seem to produce better reward
- But the curves do not seem to reach an asymptotic behaviour, it is continually oscillating

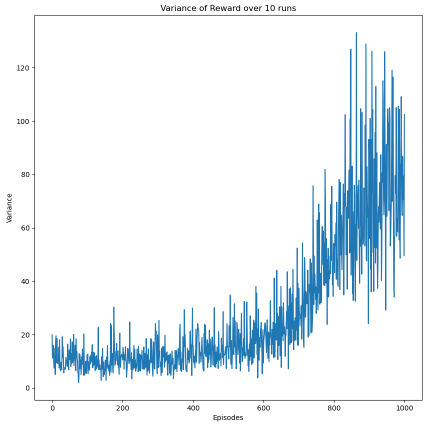
5-Step AC Experiment 3

Hyperparameters

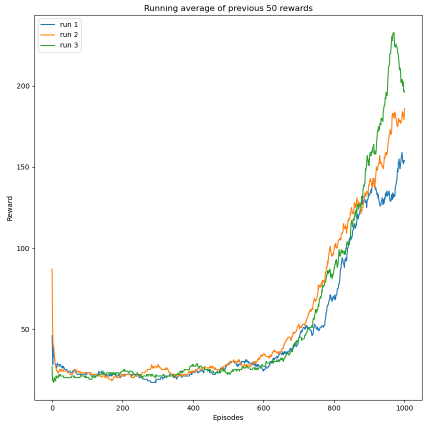
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 0.5 * 10^{-4}$

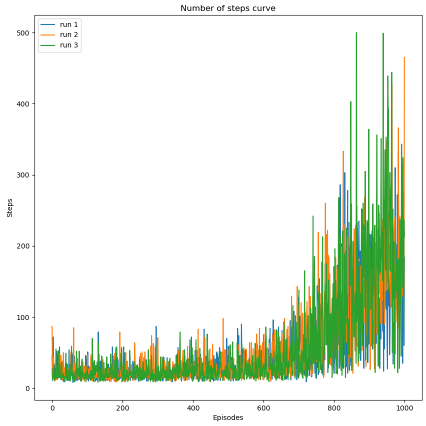
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

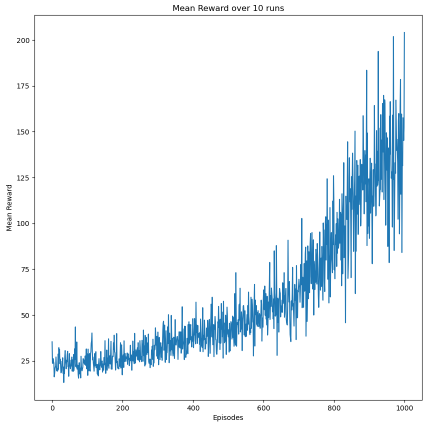
- With decrease in learning rate, the agent seems to perform much better than the previous two experiments.
- Lowering learning rate has helped in this, without occurence of overshooting behaviour

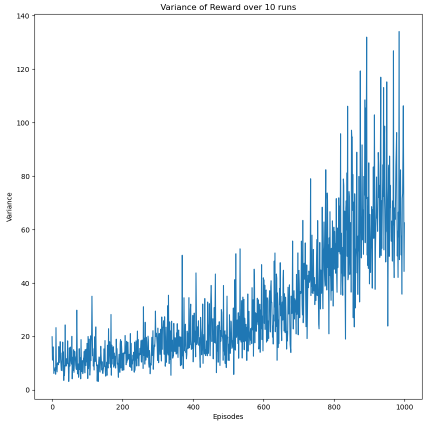
10-Step AC Experiment 1

Hyperparameters

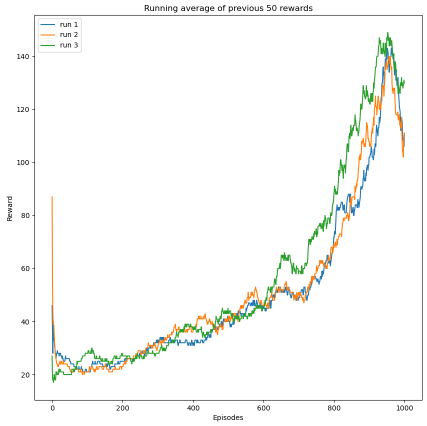
- hidden layer count $= 1$
- hidden layer sizes $= [1024]$
- learning rate $\alpha = 1 * 10^{-4}$

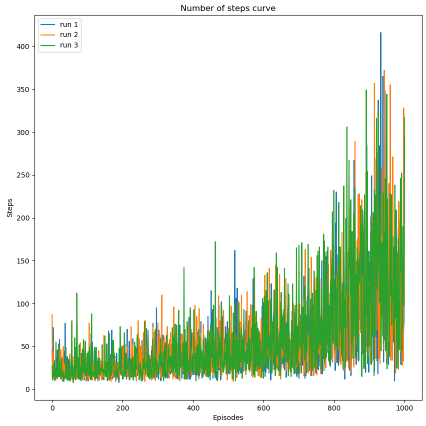
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

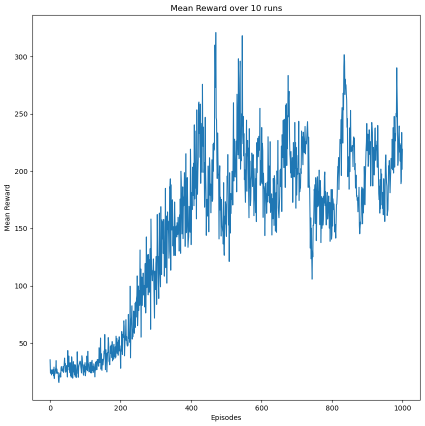
- Very slow convergence leading to a very slow increase in the reward curve for this network architecture
- Variance is also very low compared to one step actor critic experiment

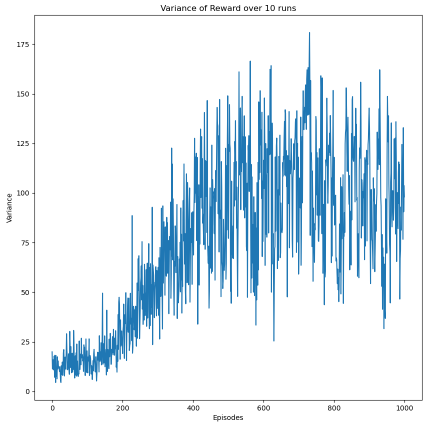
10-Step AC Experiment 2

Hyperparameters

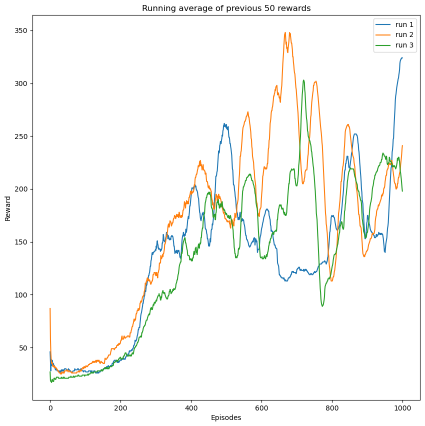
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

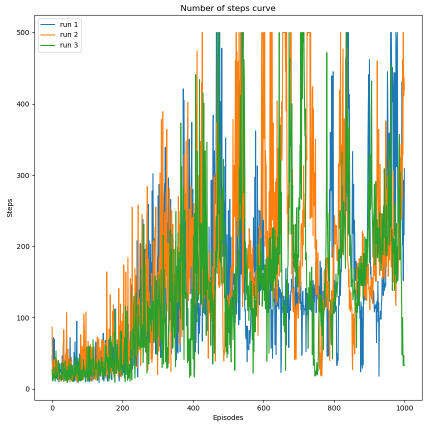
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

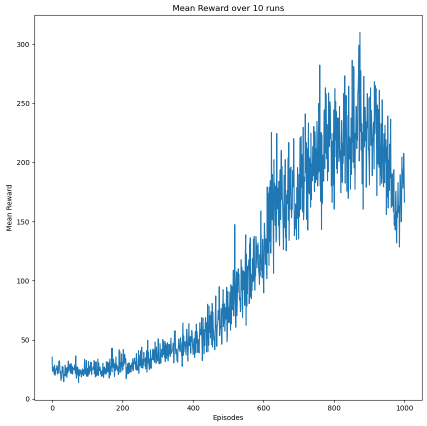
- Increase in size of network architecture did not bring any improve the performance
- Variance is also very low compared to one step actor critic experiment

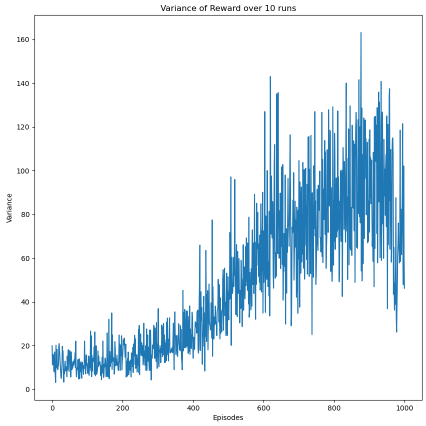
10-Step AC Experiment 3

Hyperparameters

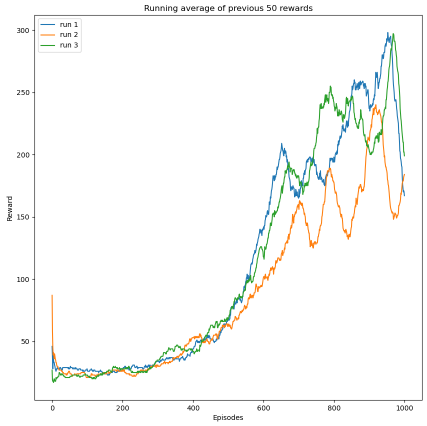
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 0.5 * 10^{-4}$

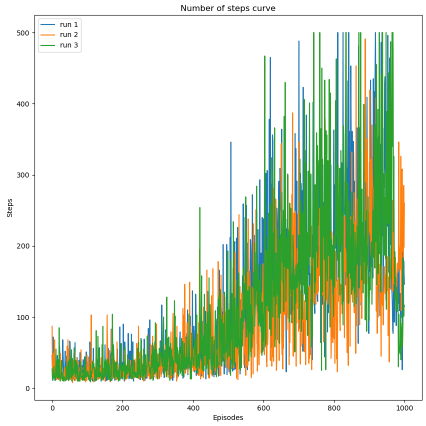
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

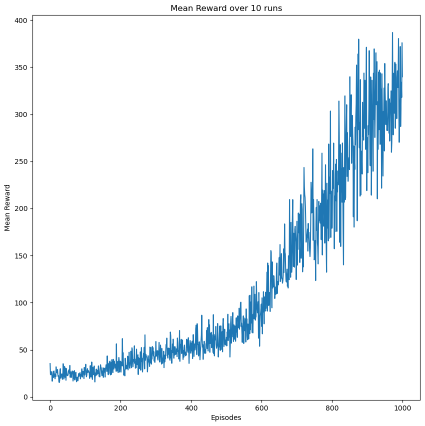
- For the fixed architecture, reducing learning rate has increased the performance of the agent
- Even though there is the presence of high variance, the learning is in increasing trend but slowly, due to smaller learning rate

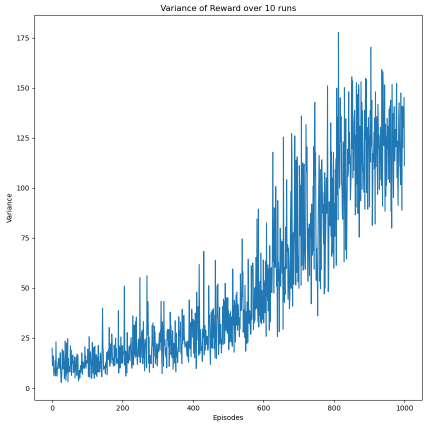
Full Return AC Experiment 1

Hyperparameters

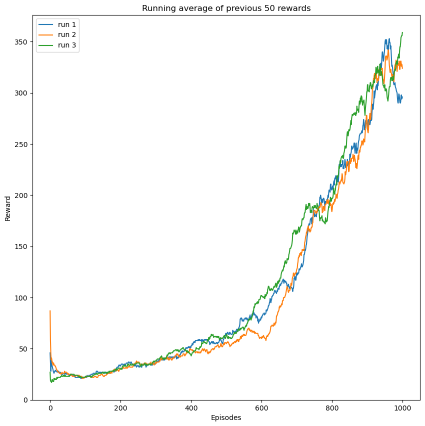
- hidden layer count $= 1$
- hidden layer sizes $= [1024]$
- learning rate $\alpha = 1 * 10^{-4}$

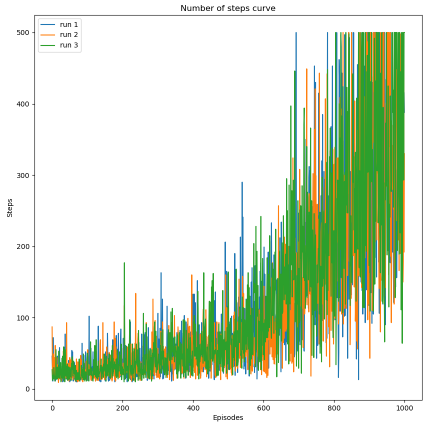
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

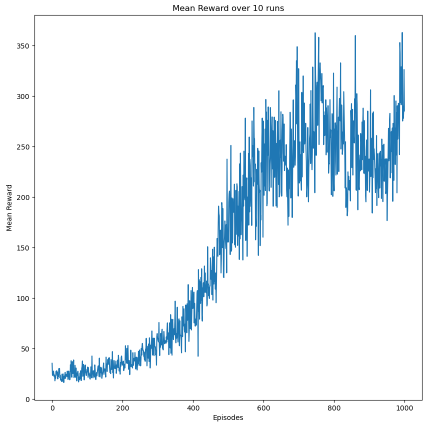
- Mean reward is strictly increasing with less variance
- This set of hyperparameters seem to produce maximum achievable optimality in the policy

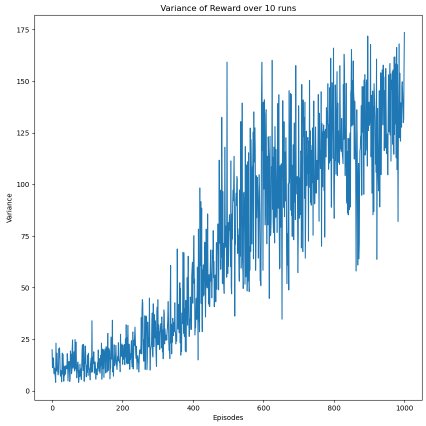
Full Return AC Experiment 2

Hyperparameters

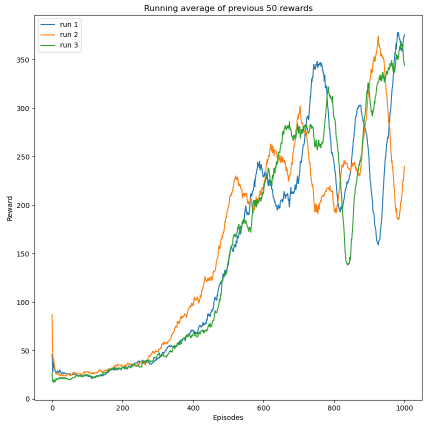
- hidden layer count $= 2$
- hidden layer sizes $= [512, 512]$
- learning rate $\alpha = 0.5 * 10^{-4}$

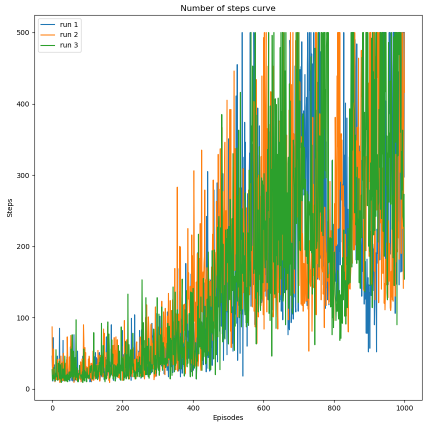
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

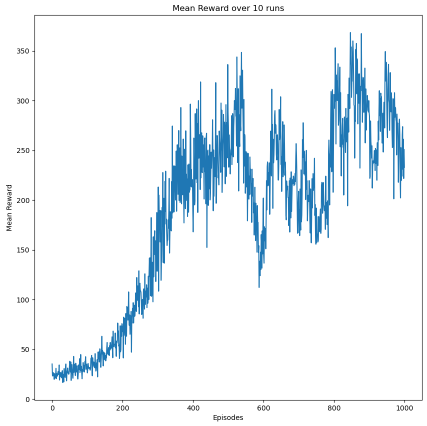
- This behaviour seems unpredictable, neither lowering learning rate nor increase network size seems to increase the performance

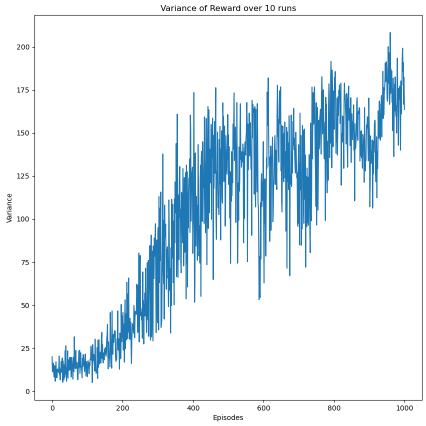
Full Return AC Experiment 3

Hyperparameters

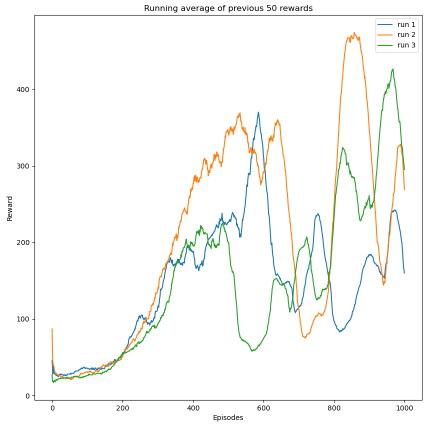
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 1024]$
- learning rate $\alpha = 0.5 * 10^{-4}$

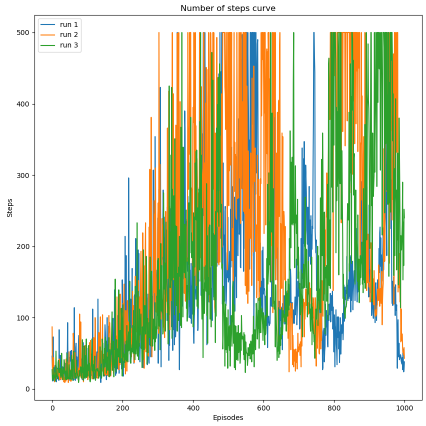
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification

- Again this behaviour seems unpredictable, neither lowering learning rate nor increase network size seems to increase the performance
- The architecture size is larger than experiment 2, and the performance is even more bad with high variance and unstable reward learning curve

Overall Comments for Cartpole environment

- 1-step, 5-step and 10-step results were all similar with respect to number of steps in achieving the cartpole stability
- Variation in variance also occured similarly across the AC variations with changes in the architecture and learning rate
- Full return Actor critic method had the best outputs of all the four variations with increasing and stabilizing reward curves but took computationally more time to run

MountainCar-v0

General Technique

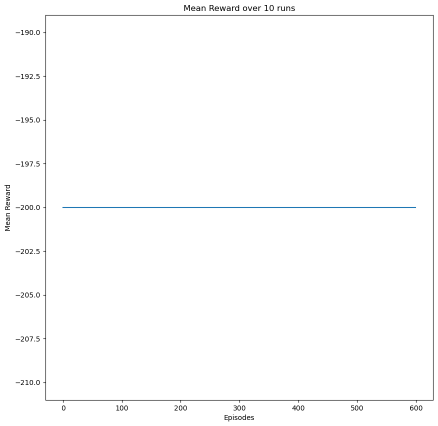
- We followed the same technique as the previous environments here as well.
- However, the agent did not achieve episode reward > -200 even once.
- Our justification and inferences are given after each experiment.

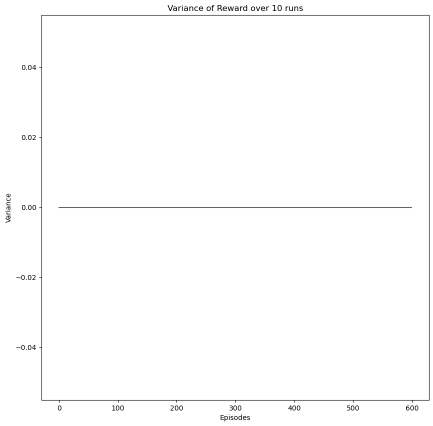
1 Step AC Experiment

Hyperparameters

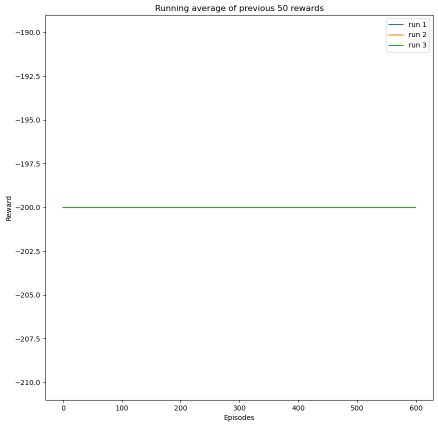
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

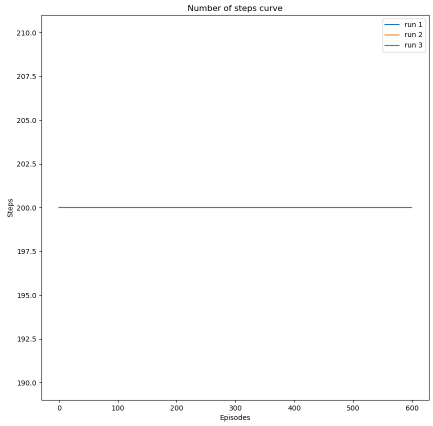
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





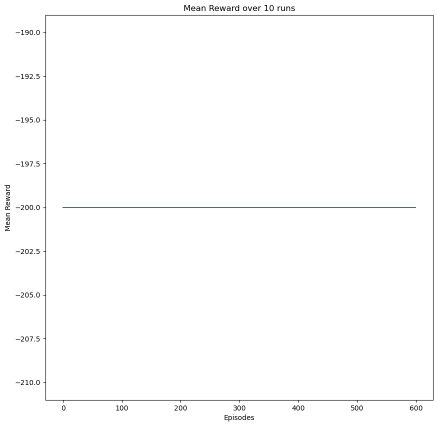
Justification/Inferences The agent fails to learn anything (achieve an episode reward ¿ -200) in this environment.

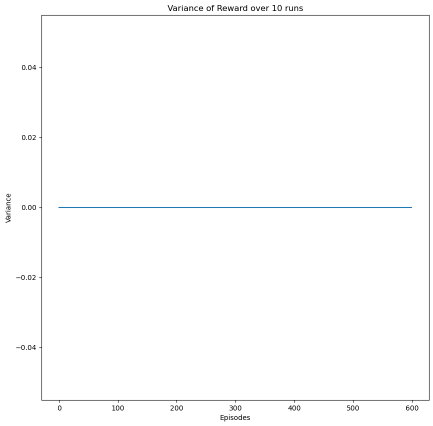
5-Step AC Experiment

Hyperparameters

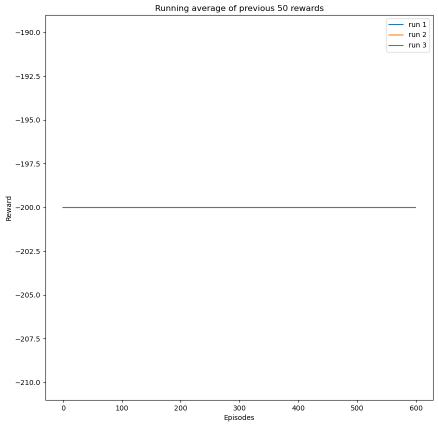
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

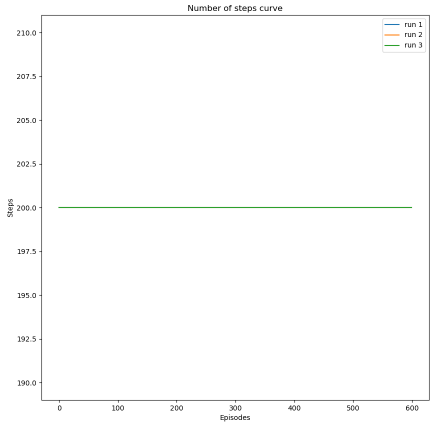
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





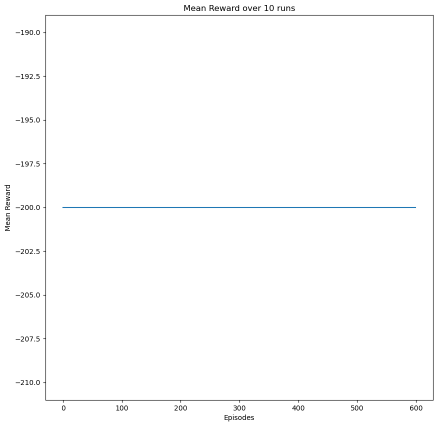
Justification/Inferences The agent fails to learn anything (achieve an episode reward ¿ -200) in this environment.

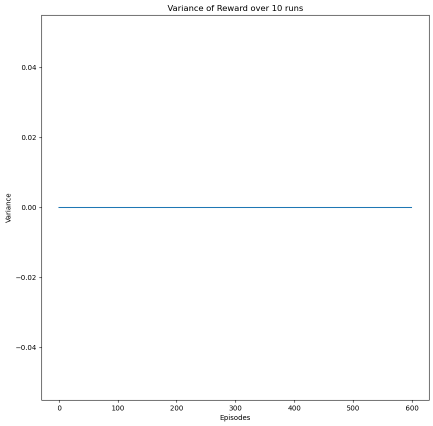
10-Step AC Experiment

Hyperparameters

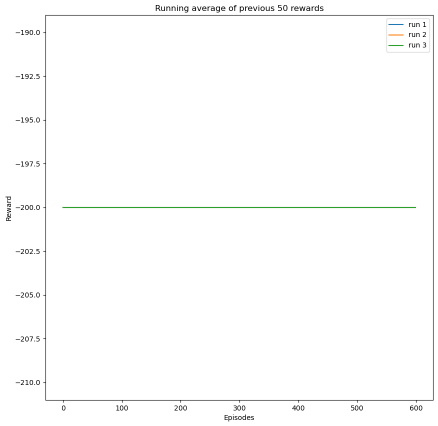
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

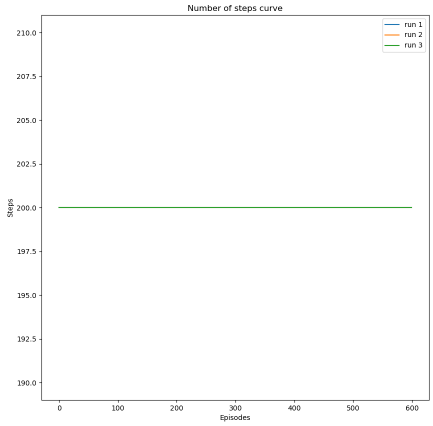
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





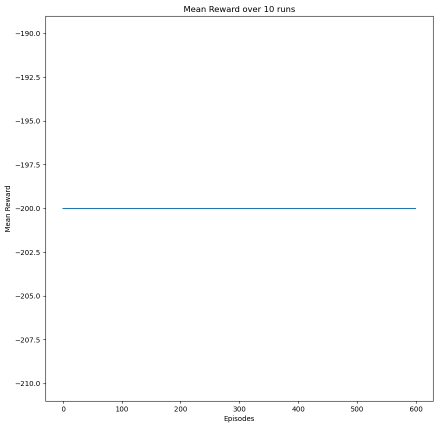
Justification/Inferences The agent fails to learn anything (achieve an episode reward ¿ -200) in this environment.

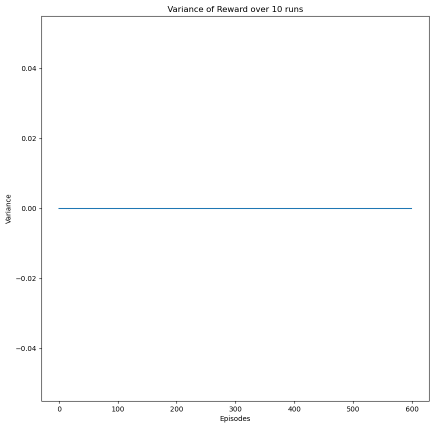
Full Return AC Experiment

Hyperparameters

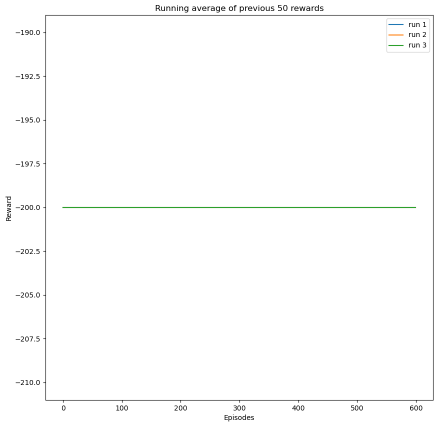
- hidden layer count $= 2$
- hidden layer sizes $= [1024, 512]$
- learning rate $\alpha = 1 * 10^{-4}$

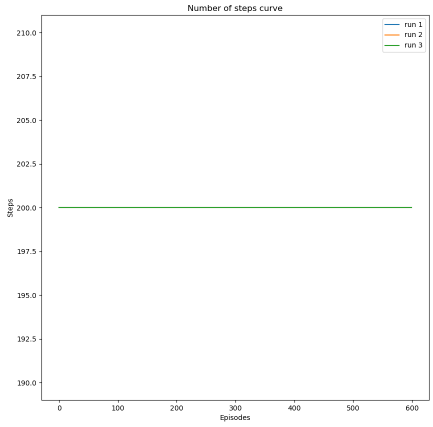
Mean Reward Curve and Variance Curve for the 10 runs





Reward Curve and Steps Curve for the first 3 runs started





Justification/Inferences The agent fails to learn anything (achieve an episode reward ¿ -200) in this environment.

Overall Comments for MountainCar environment

- The agent does not learn anything for all AC variations
- Tuning hyperparameters does not have any effect in the improvement of the performance

Conclusions for Actor-critic methods

- Despite our many experiments with network architecture and learning rate, we could not get the agent to reach the goal in any episode. This can be seen from the constant negative reward of -200.
- We believe this is due to lack of exploration by AC methods, sparse and non-informative rewards provided by the mountain car environment. The constant negative reward of -1 for all timesteps when goal is not reached does not help in directing the agent to the goal.
- We believe the non-exploratory behaviour of AC methods can be rectified by sampling more exploratively, and using eligibility traces to distribute reward effect to past states' values. Also, taking noisy actions may also help with exploration.
- Possibly, doing reward engineering more carefully will help with the sparse and non-informative reward issue.

Link to all the plots and rendered outputs(gifs) for different experiments: DQN,AC
Link to all the notebooks that we used to produce these results : Notebooks