CS6700 Reinforcement Learning
Report for PA #2

Varun Gumma CS21M070
Harsha Vardhan Gudivada CS21M021

1 Introduction

In this assignment, we experimented with 3 different `OpenAI Gym` Environments, namely, `CartPole-v1`, `Acrobot-v1` and `MountainCar-v0`. For each environment we find 5 hyperparameter sets such that $S_1 \geq S_2 \geq S_3 \geq S_3 \geq S_4 \geq S_5$. For mountain-car, we find most of the curve to be flat-lines, this is because the agent only receives the reward when it reaches the flag. Till then, it gets no rewards and oscillates significantly and hence the reward is stuck at -200 (-1 for each step).

2 Deep Q-Network

2.1 CartPole-v1

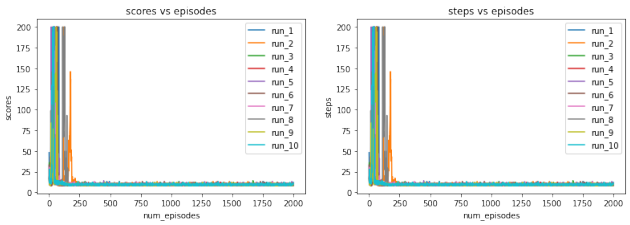


Figure 1: Average reward: 11.58. batch_size: 128, buffer_size: 1000000, epsilon_decay: 0.9, gamma: 0.999, lr: 0.0005, num_hidden_units: 128, update_freq: 20

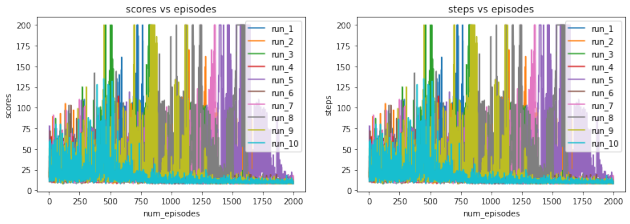


Figure 2: Average reward: 23.84. batch_size: 128, buffer_size: 100000, epsilon_decay: 0.999, gamma: 0.999, lr: 0.01, num_hidden_units: 256, update_freq: 20

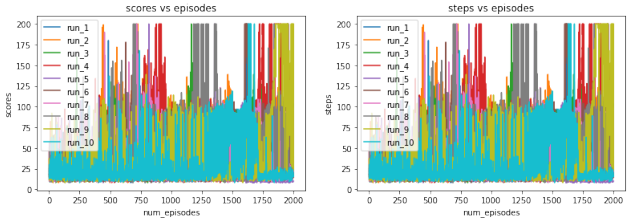


Figure 3: Average reward: 35.42. batch_size: 512, buffer_size: 1000000, epsilon_decay: 0.999, gamma: 0.99, lr: 0.01, num_hidden_units: 128, update_freq: 75

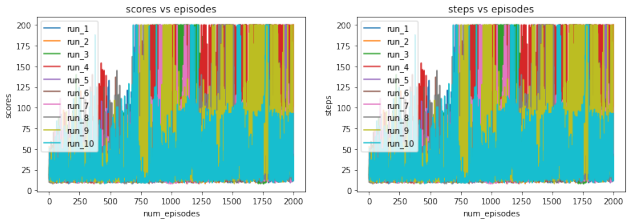


Figure 4: Average reward: 45.27. batch_size: 128, buffer_size: 1000000, epsilon_decay: 0.999, gamma: 0.99, lr: 0.0005, num_hidden_units: 512, update_freq: 50

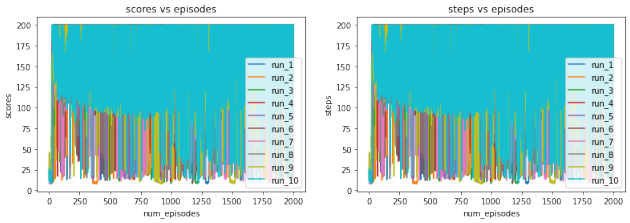


Figure 5: Average reward: 141.69. batch_size: 256, buffer_size: 100000, epsilon_decay: 0.9, gamma: 0.9, lr: 0.001, num_hidden_units: 256, update_freq: 20

2.2 Acrobot-v1

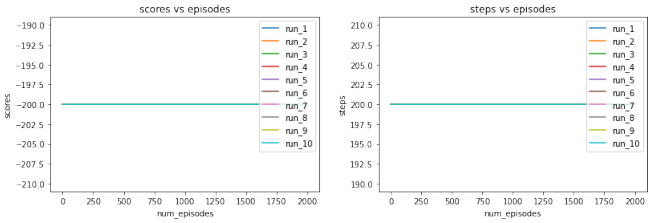


Figure 6: Average reward: -195. batch_size: 128, buffer_size: 100000, epsilon_decay: 0.999, gamma: 0.9, lr: 0.01, num_hidden_units: 512, update_freq: 75

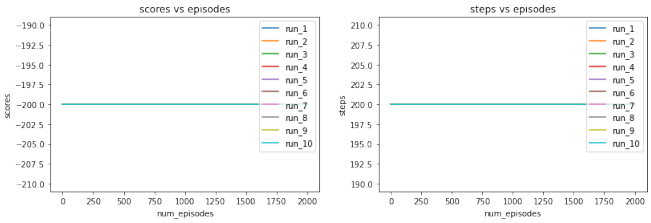


Figure 7: Average reward: -200. batch_size: 128, buffer_size: 100000, epsilon_decay: 0.99, gamma: 0.9, lr: 0.01, num_hidden_units: 128, update_freq: 75

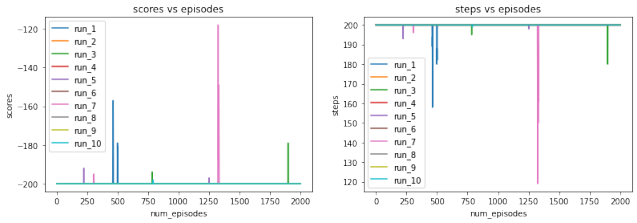


Figure 8: Average reward: -200. batch_size: 256, buffer_size: 1000000, epsilon_decay: 0.999, gamma: 0.999, lr: 0.001, num_hidden_units: 512, update_freq: 40

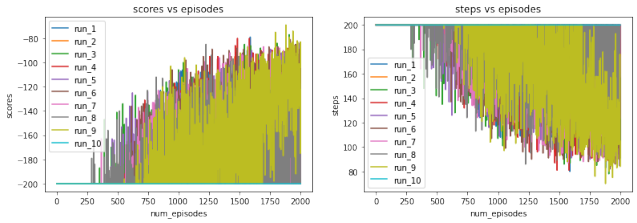


Figure 9: Average reward: -182.77. batch_size: 512, buffer_size: 1000000, epsilon_decay: 0.999, gamma: 0.9, lr: 0.001, num_hidden_units: 128, update_freq: 20

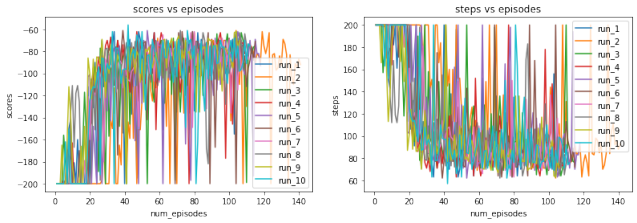


Figure 10: Average reward: -99.5. batch_size: 256, buffer_size: 1000000, epsilon_decay: 0.9, gamma: 0.999, lr: 0.0005, num_hidden_units: 128, update_freq: 20

2.3 MountainCar-v0

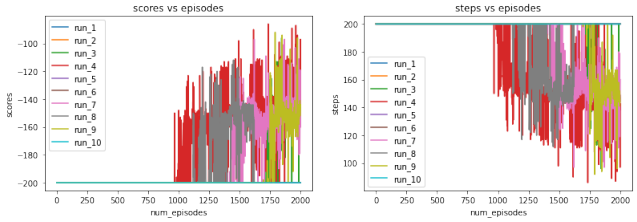


Figure 11: Average reward: -194.24. batch_size: 512, buffer_size: 100000, epsilon_decay: 0.999, gamma: 0.99, lr: 0.001, num_hidden_units: 128, update_freq: 100

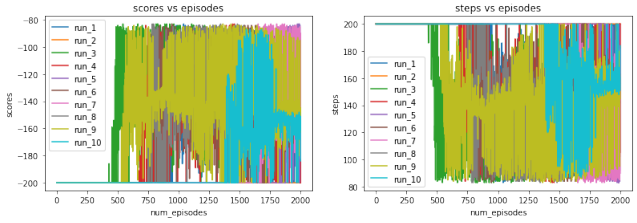


Figure 12: Average reward: -157.28. batch_size: 256, buffer_size: 1000000, epsilon_decay: 0.9, gamma: 0.99, lr: 0.0005, num_hidden_units: 256, update_freq: 20

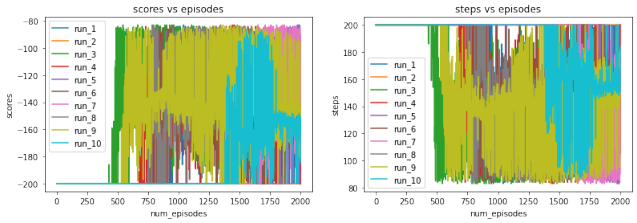


Figure 13: Average reward: -137.51. batch_size: 512, buffer_size: 100000, epsilon_decay: 0.99, gamma: 0.99, lr: 0.0005, num_hidden_units: 128, update_freq: 20

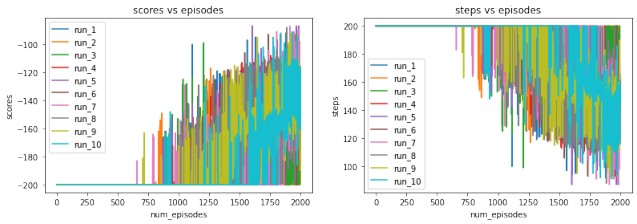


Figure 14: Average reward: -133.56. batch_size: 256, epsilon_decay: 0.999, gamma: 0.99, lr: 0.01, num_hidden_units: 512, update_freq: 15

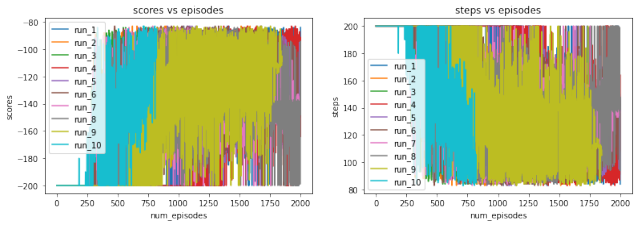


Figure 15: Average reward: -109.60. batch_size: 128, epsilon_decay: 0.99, gamma: 0.99, lr: 0.0005, num_hidden_units: 128, update_freq: 10

2.4 Observations

1. For each environment, we terminate the current run if the average of the last 100 episodes exceeds the value given in `env.spec.reward_threshold`.

2. Lower learning rates are preferred. This is because, DQN models by itself are very stochastic and do not have a fixed target (target dynamically changes). A higher learning rate, makes larger steps end is more prone not to converge.

3. Buffer size (above 10^5) does not matter much. As we do not want a some particular set of experiences to used always, we maintain experiences from several timesteps. Once a sufficiently large number of experiences are stored, the probability of the same batch being sampled reduces exponentially. Also, in the initial stages of learning by the agent (when the buffer has almost batch size number of elements) then the similar set of experiences get sampled over-and-over again and the training might be very slow and not so progressive. Once enough experiences are accumulated, the learning *jump starts*. With 2000 episodes and 200 steps per episodes, we accumulate atmost $4 \cdot 10^5$ experience and if we have a buffer of size more than that, we will never discard any of the previous experiences. This is equivalent to storing all experiences, even the inital ones which might be very relevant at the current timestep.

4. With larger update frequency, the training is worse as there will be a large disparity between the target and local networks. But in reality, we want the target and local networks to be as close as possible and *mimic* each other. This also increases the running time.

5. Batch size also was found not to influence the training much. Higher batch sizes have higher computation time. But a higher batch size also helps us sample more experiences (which may involve a few older ones as well). This mitigates the issue of *catastrophic forgetting*. Batch sizes of 128, 256 performed better than 16, 32.

6. Gradient Clamping helps in convergence. Since we have a moving target, it is better to take smaller steps during gradient descent to avoid over-shooting/bouncing around the minima. Hence, along with lower learning rate, we clamp the gradient values to $(-5, 5)$.

7. Atleast 128 hidden nodes are needed in both layers for proper convergence. Since `Acrobot-v1` and `MountainCar-v0` are more complex environments than `CartPole-v1`, a bigger neural network was required for them. Our very initial attempts with 10-15 hidden units did not improve the value at all.

8. Changes to gamma did not improve/impair the learning. We checked very high values of gamma and came to this conclusion. Lower values will have more discounting and will provide lower rewards.

9. Higher values of epsilon-decay were favourable. This was because, till the time the buffer had atleast batch size number of experiences, the agent was acting randomly and and its epsilon value still decayed, i.e. if the batch size was b, by the time the NN actually starts performing $\epsilon_t = \delta^b$ (where δ is the epsilon-decay-rate and $\epsilon_0 = 1$). Even of the decay rate was 0.99 and batch size was 64, $\epsilon_{64} = 0.5255$ which implies the has already started preferring some current maximum 48% of the time before any proper training has begun.

10. We preferred HuberLoss for the network to MSE. This was because, by using mean-average-error for outliers in the training, we penalize then a bit lower than with mean-square-error, and we know RL agent training has a lot of fluctuations and jitter. Though upon comparison, we did not find much improvement. Both trained similarly.

11. Changing optimizer did not influence the training much. `Adam`, `SGD` (with nesterov), `RMSprop` behaved similarly. Hence, we stuck to the default `Adam`. But `SGD` was found to be computationally easy and very slightly improved the runtime.

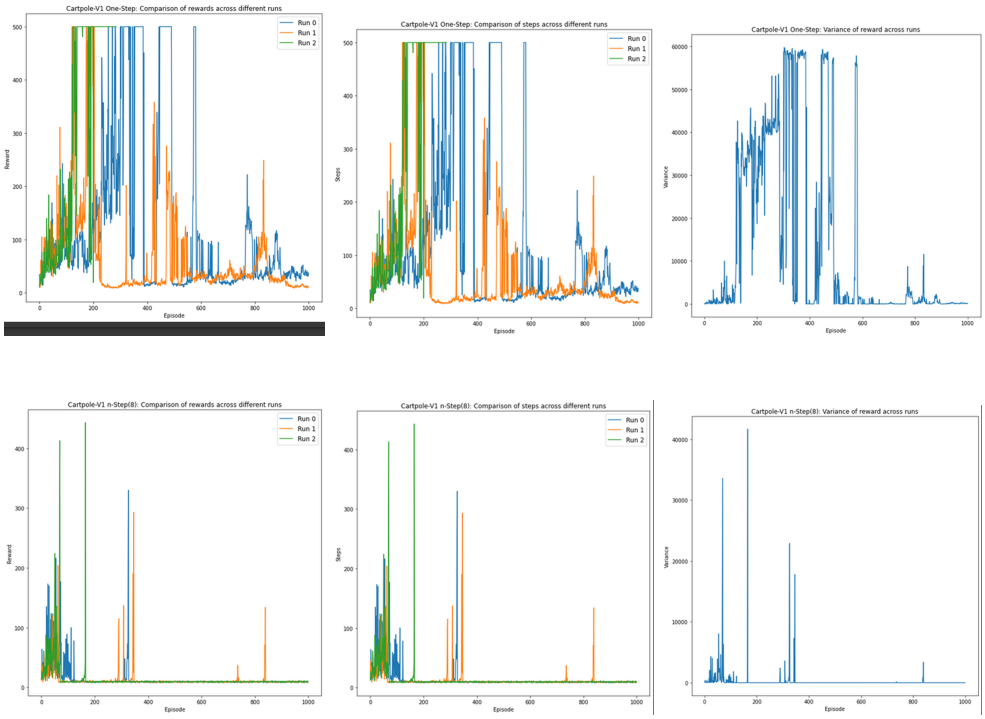
12. Similarly, changing the hidden activation to other `ReLU` derivatives like `selu`, `elu`, `swish` and `softplus` did not influence the training much.

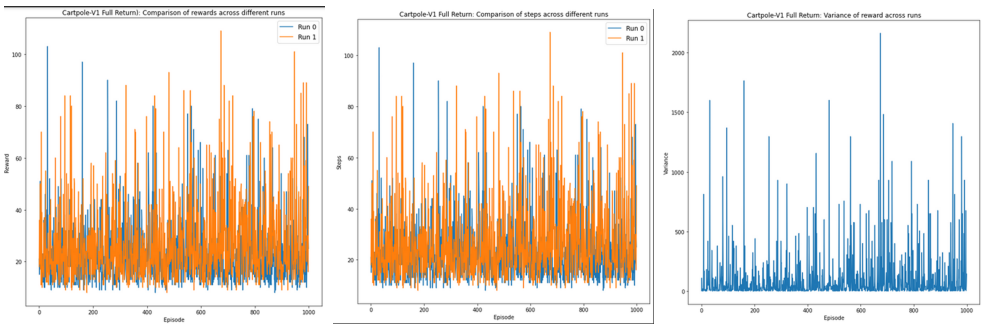
13. Increasing the hidden layers more than two only increased the computation but did not produce any better results and hence we set it to two only.

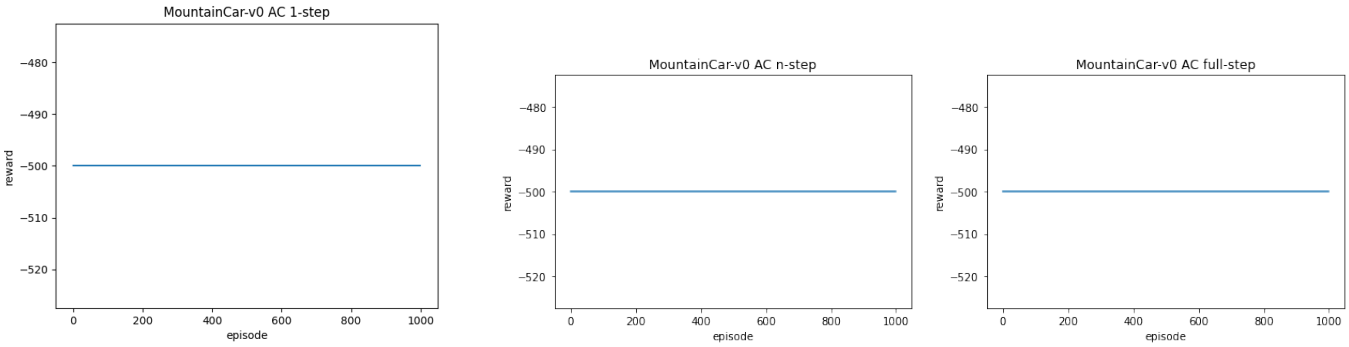
14. The hyperparameter configurations were tested with the help of `Wandb Bayes Search`.

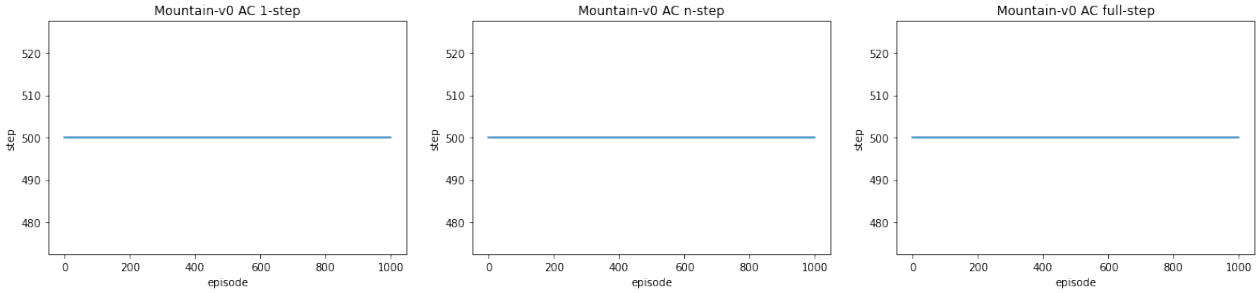
3 Actor-Critic Model

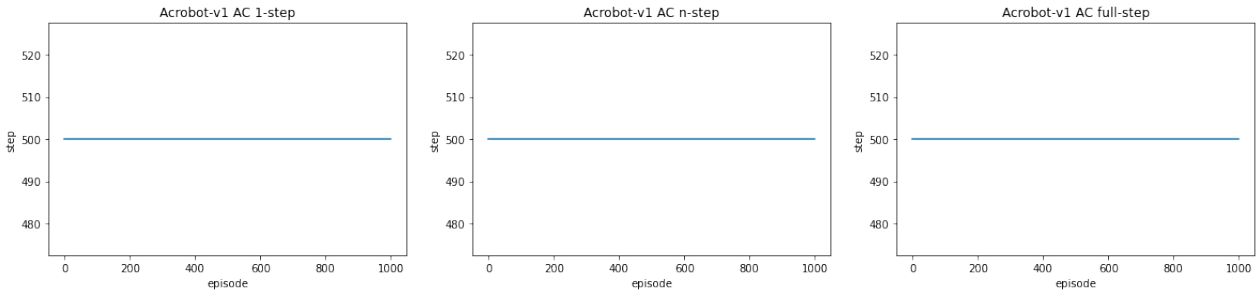
3.1 Plots

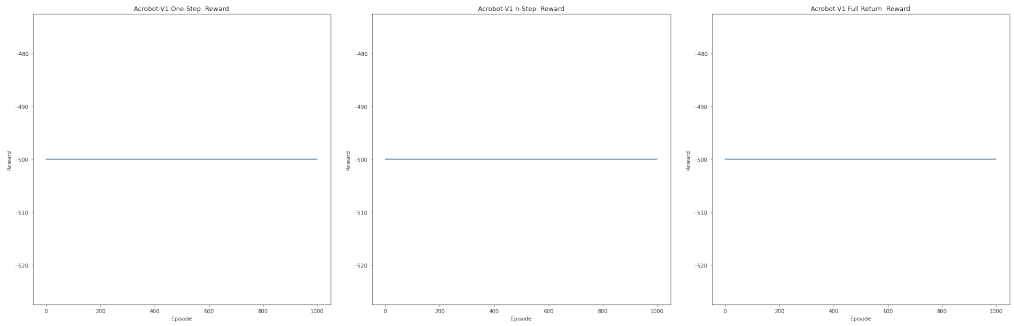



Cartpole-V1 Full Return: Comparison of rewards across different runs
Run 0
Run 1
Reward
Episode
Cartpole-V1 Full Return: Comparison of steps across different runs
Run 0
Run 1
Steps
Episode
Cartpole-V1 Full Return: Variance of reward across runs
Variance
Episode


MountainCar-v0 AC 1-step
reward
episode
MountainCar-v0 AC n-step
reward
episode
MountainCar-v0 AC full-step
reward
episode


Mountain-v0 AC 1-step
step
episode
Mountain-v0 AC n-step
step
episode
Mountain-v0 AC full-step
step
episode


Acrobot-v1 AC 1-step
step
episode
Acrobot-v1 AC n-step
step
episode
Acrobot-v1 AC full-step
step
episode


Acrobot-V1 One-Step: Reward
Reward
Episode
Acrobot-V1 n-Step: Reward
Reward
Episode
Acrobot-V1 Full Return: Reward
Reward
Episode

3.2 Observations

1. 3 variations of actor critic methods - one-step, n-step and full return are used for the experiments.
2. Some plots are shown there. Other environment's good hyper parameters could not be found. As there is no change in the reward in some reward plots, their variation plots were not included.
3. Same network (same parameters) were used in all the variations in order to compare them.
4. From the plots, we can see that all the Actor Critic variations have high variance, which is a general property of any Policy Gradient Method.
5. During the tuning of hyper parameters, all the variations are very sensitive to any small change in the hyper-parameters like learning rate and hidden layer sizes.
6. Variance is reducing across the runs as we go from one-step to n-step to full-return. As per the theory, full-return has high variance across the episodes due to stochasticity of the environment and the policy. So, across the runs, full-returns variation tends to be more stable.
7. So, to have moderate variance and bias between both one-step and full-return, n-step is helpful. However, in this assignment, tuning the 'n' value did not change the performance much.