CS5691: Reinforcement Learning
Assignment #2

Topics: DQN,Policy Gradient **Deadline:** 30 March 2022

Teammate 1: (AKASH SAINI) **Roll number:** CS21M003
Teammate 2: (RETURAJ BURNWAL) **Roll number:** CS21D406

- Code link: Colab Notebook
- **DQN**

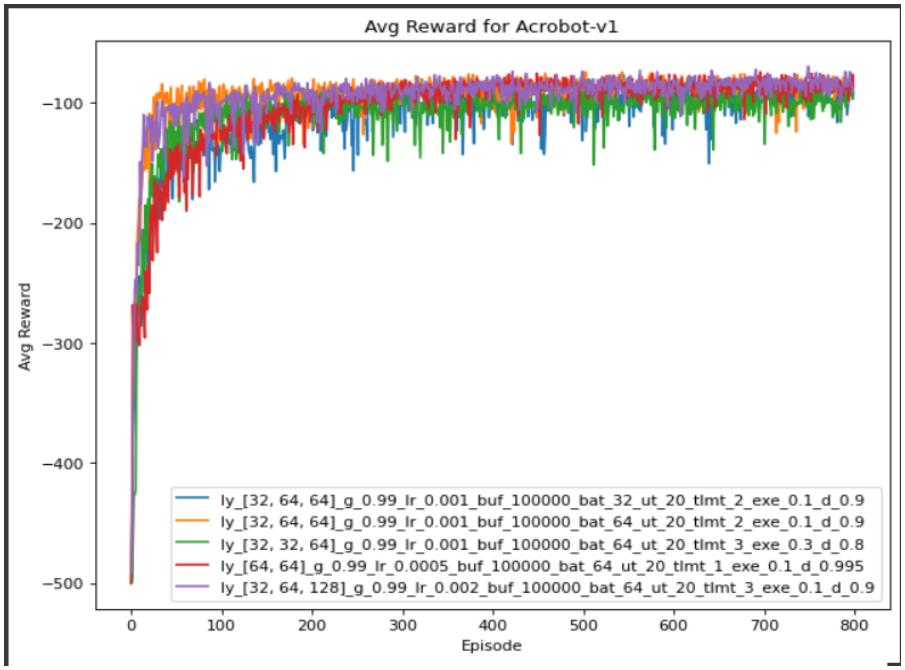
Solution:

Metric chosen: We are measuring the performance by comparing average reward value. Those hyper-parameters that gave higher average reward value with less number of episodes are considered better.

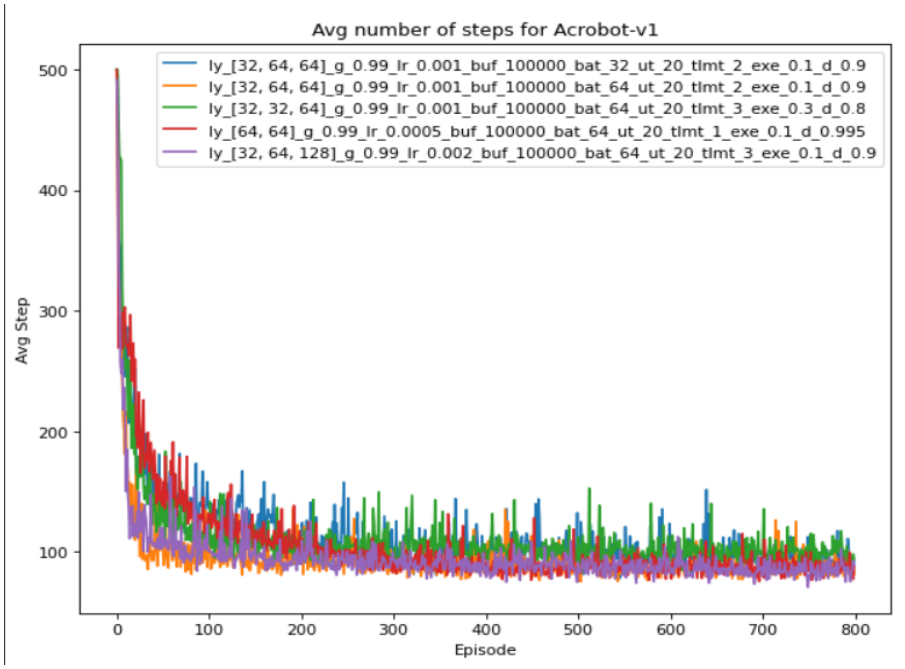
Exploration Algorithm: We used softmax algorithm.

Define:
1) Layer(24, 32, 16) : means a MLP is constructed with first hidden layer has 24 nodes, second hidden layer has 32 nodes, third hidden layer has 16 nodes.

1. **Acrobot-v1**
   - Avg reward plot:



– Avg step to reach goal plot:



– Observations and Explanation:

  * Hyper-parameter: Layer(32, 32, 64), gamma=0.99, learning_rate=0.001, buffer=10^5, batch=64, update_target=20, truncation_limit=3, β_{end}=0.3,

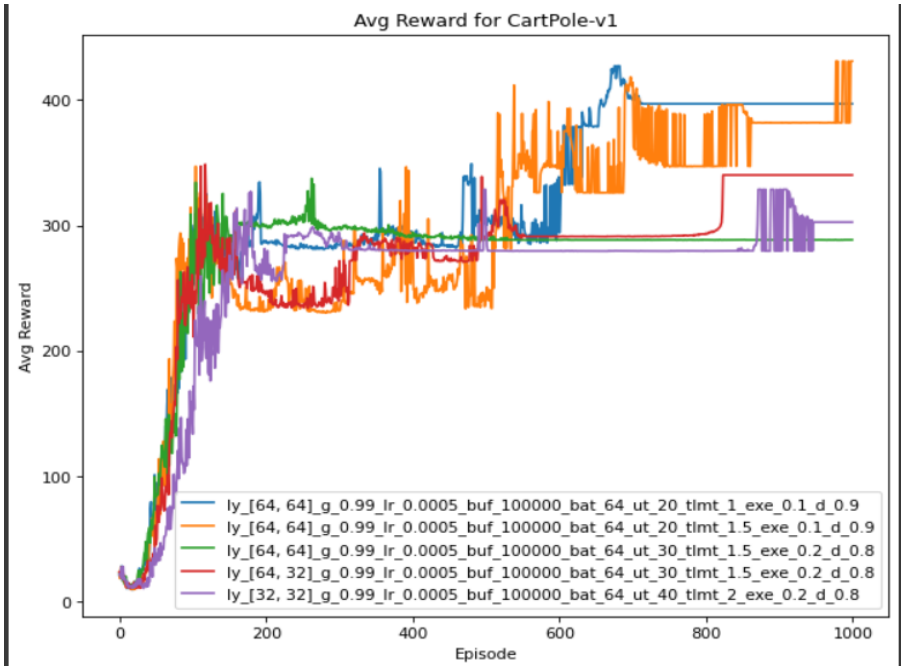
β_{decay}=0.8

- This set of hyper-parameters performs relatively poor.
- We observed that increasing the model complexity helps in improving the performance.
- We also found that increasing the learning rate and truncation limit helps model to make larger updates to its weights. This in turn helped model to learn faster.

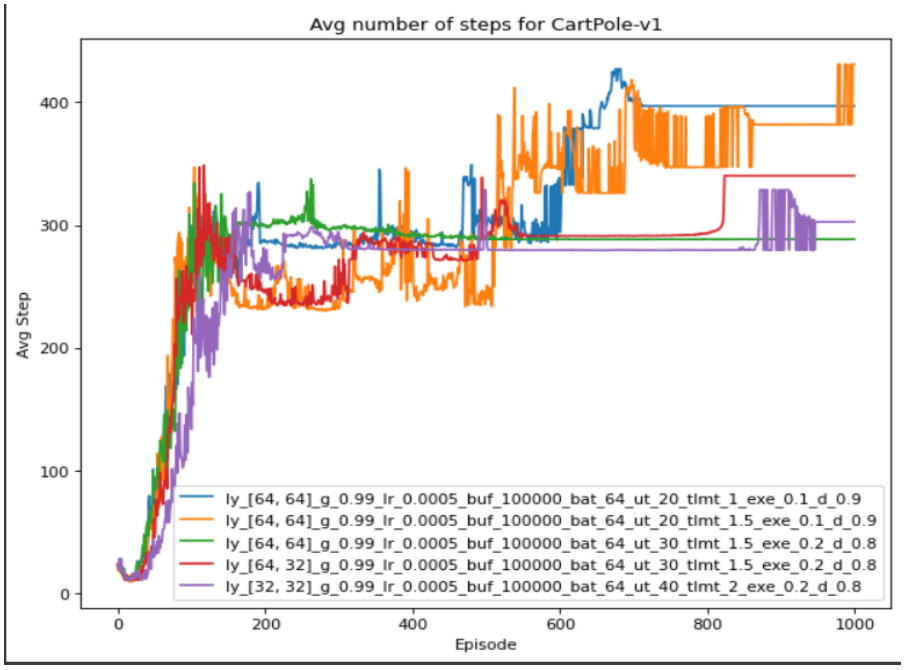
* Hyper-parameter: Layer(32, 32, 64), gamma=0.99, learning_rate=0.001, buffer=10^5, batch=32, update_target=20, truncation_limit=2, β_{end}=0.3, β_{decay}=0.9
  - This set of hyper-parameters performs relatively better than above hyper-parameter.
  - We found that decaying beta slowly helps model to explore more initially and in turn helps model to generalize well for unseen states.
* Hyper-parameter: Layer(64, 64), gamma=0.99, learning_rate=0.0005, buffer=10^5, batch=64, update_target=20, truncation_limit=1, β_{end}=0.1, β_{decay}=0.995
  - This set of hyper-parameters performs relatively better than above hyper-parameters.
  - We tried to increase the model complexity by increasing the number of nodes in the initial layer.
  - Even though the learning rate and truncation limit is low, we were able to achieve good performance. We believe that this performance is due to more initial exploration and more exploitation in the end.
* Hyper-parameter: Layer(32, 64, 64), gamma=0.99, learning_rate=0.001, buffer=10^5, batch=64, update_target=20, truncation_limit=2, β_{end}=0.1, β_{decay}=0.9
  - This set of hyper-parameters performs relatively better than above hyper-parameters.
  - Based on the previous attempts on improving the performance: we decided to increase the model complexity, choose higher learning rate and truncation limit and set higher beta decay and lower beta end value.
* Hyper-parameter: Layer(32, 64, 128), gamma=0.99, learning_rate=0.002, buffer=10^5, batch=64, update_target=20, truncation_limit=3, β_{end}=0.1, β_{decay}=0.9
  - This set of hyper-parameters performs the best among all the experiments that we performed.
  - We further increased the model complexity with higher learning rate and higher truncation limit value.

2. **CartPole-v1**

– Avg reward plot:



– Avg step to reach goal plot:

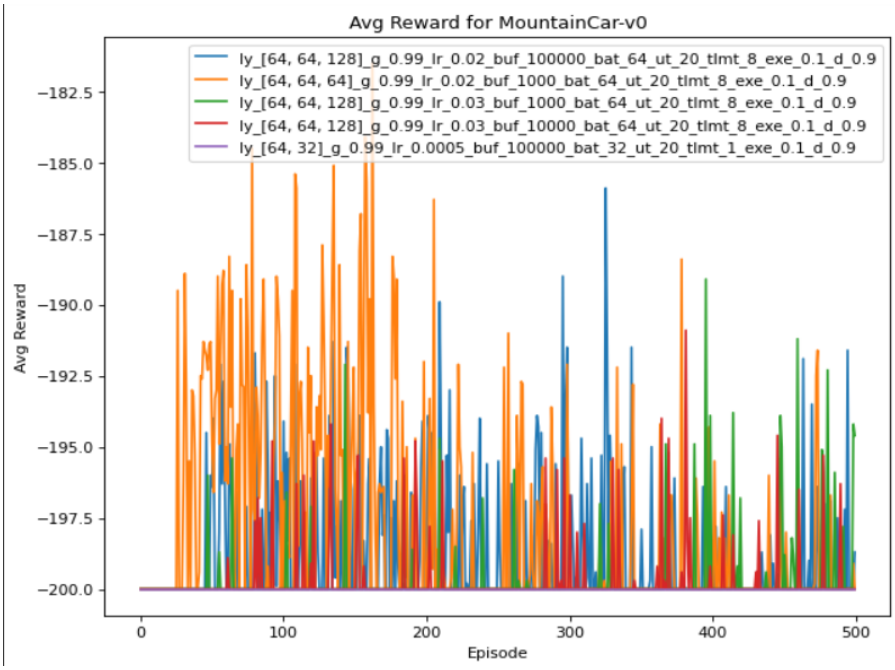


– Observations and Explanation:

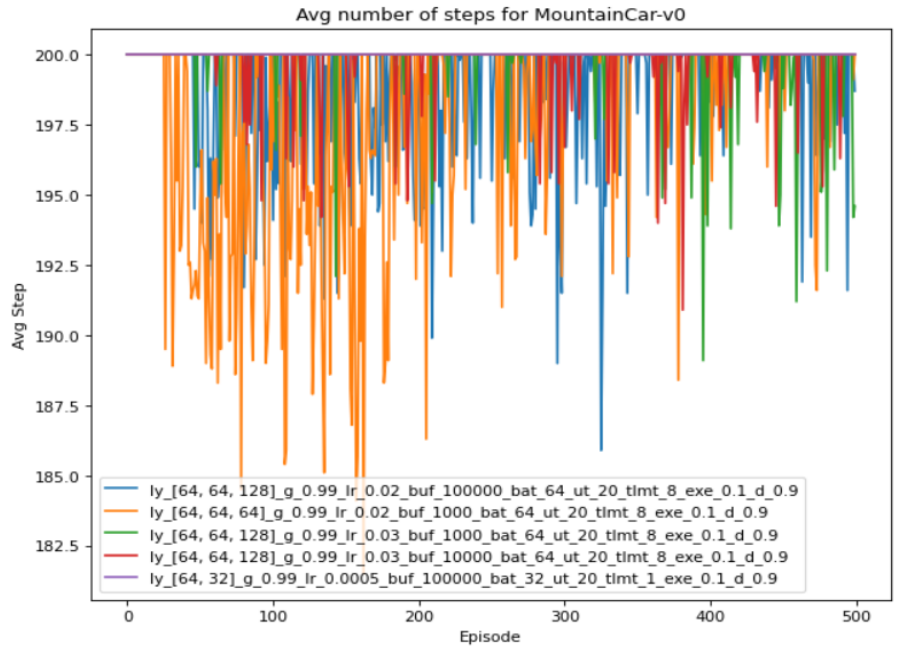
* Hyper-parameter: Layer(64, 64), gamma=0.99, learning_rate=0.0005, buffer=10^5, batch=64, update_target=30, truncation_limit=1.5, β_{end}=0.2, β_{decay}=0.8
  - This set of hyper-parameters performs relatively poor, but its performance is better than the tutorial hyper-parameter.
  - Compared to tutorial hyper-parameter reducing the model complexity, increasing the truncation limit and decaying beta faster helped the model learn faster.
* Hyper-parameter: Layer(32, 32), gamma=0.99, learning_rate=0.001, buffer=10^5, batch=32, update_target=40, truncation_limit=2, β_{end}=0.2, β_{decay}=0.8
  - This set of hyper-parameters performs relatively better than above hyper-parameter.
  - We found that initially the performance is lower compared to above hyper-parameter but its gradually improved.
* Hyper-parameter: Layer(64, 32), gamma=0.99, learning_rate=0.0005, buffer=10^5, batch=64, update_target=30, truncation_limit=1.5, β_{end}=0.2, β_{decay}=0.8
  - This set of hyper-parameters performs relatively better than above hyper-parameters.
  - We tried to explore with making the model simpler and we found that it outperforms the above hyper-parameters model.
* Hyper-parameter: Layer(64, 64), gamma=0.99, learning_rate=0.0005, buffer=10^5, batch=64, update_target=20, truncation_limit=1, β_{end}=0.1, β_{decay}=0.9
  - This set of hyper-parameters performs relatively better than above hyper-parameters.
  - We found that making model explore more initially (high decay) and exploit more afterwards (low beta end) helped model generalize well and hence eventually learned better weights.
* Hyper-parameter: Layer(64, 64), gamma=0.99, learning_rate=0.0005, buffer=10^5, batch=64, update_target=20, truncation_limit=1.5, β_{end}=0.1, β_{decay}=0.9
  - This set of hyper-parameters performs the best among all the experiments that we performed.
  - Compared to 4th best model we slightly increased the truncation limit, because of which model weights get updated by slightly higher gradient values.

3. **MountainCar-v0**

   – Avg reward plot:



– Avg step to reach goal plot:



– Observation and Experiments:

  * Of all the hyper-parameter we experimented except for the following hyper-parameters, almost all gave a constant average reward of -200.

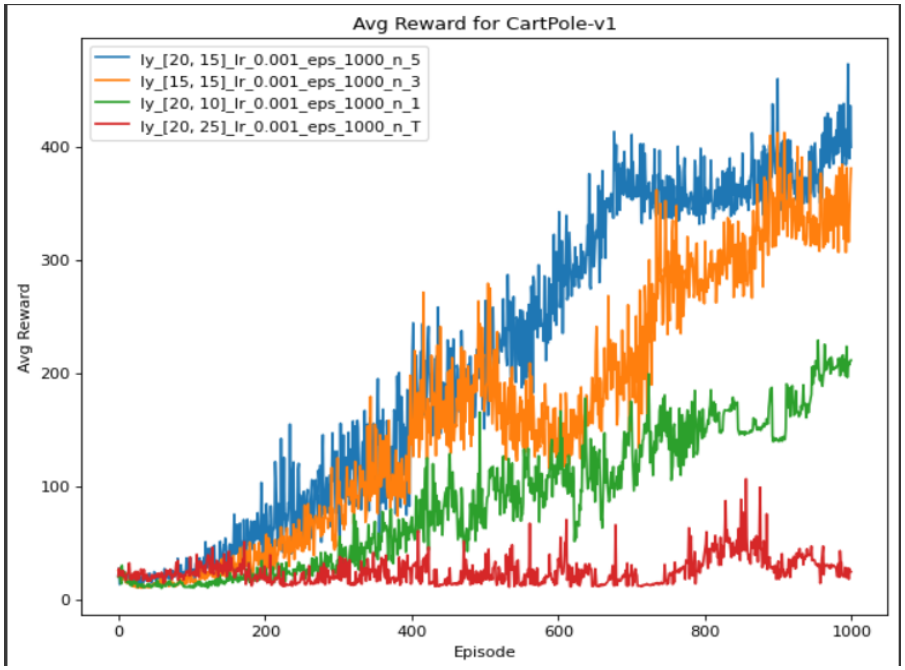
* Even after averaging with 10 runs we were not able to generate a smooth plot, so that we can compare the performance.
* We trained our model with only 500 episodes. We believe that if we had better compute resource, we could have trained our model for large number of episodes and these hyper-parameters could have learned good policy.
* We expect that the following 4 hyper-parameter configuration will perform better if trained with large number of episodes.
* We found that increasing the model complexity, learning rate and truncation limit had the most effect in improving the model.
* Hyper-parameter: Layer(64, 64, 128), gamma=0.99, learning_rate=0.02, buffer=10^5, batch=64, update_target=20, truncation_limit=8, β_{end}=0.1, β_{decay}=0.9
* Hyper-parameter: Layer(64, 64, 64), gamma=0.99, learning_rate=0.02, buffer=10^3, batch=64, update_target=20, truncation_limit=8, β_{end}=0.1, β_{decay}=0.9
* Hyper-parameter: Layer(64, 64, 128), gamma=0.99, learning_rate=0.03, buffer=10^3, batch=64, update_target=20, truncation_limit=8, β_{end}=0.1, β_{decay}=0.9
* Hyper-parameter: Layer(64, 64, 128), gamma=0.99, learning_rate=0.03, buffer=10^5, batch=64, update_target=20, truncation_limit=8, β_{end}=0.1, β_{decay}=0.9

- **Actor Critic Methods**

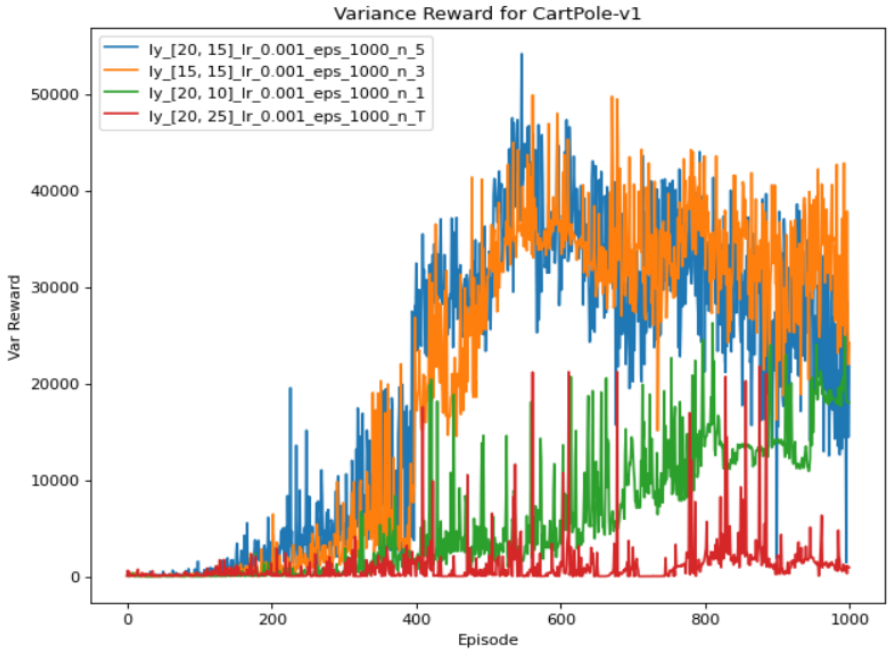
Solution:

1. **CartPole-v1**
   - Observations:
     * We found that TD(n=5) and TD(n=3) significantly outperforms TD(1) and TD(Monte carlo) methods in terms of number of steps required to reach goal state.
     * In our model we observed a high variance in the total episode reward. We were not able to observe high variance in monte-carlo method and low bias in TD(1) method.
     * But we were able to observe that Var(TD(1)) < Var(TD(n=3)) ≤ Var(TD(n=5))
     * Theoretically, we should have seen high variance in monte-carlo method, but we are not able to observe the same in our experiments. We believe that this mismatch is because of less numbers of samples (here 10 runs) are used to calculate the variance.

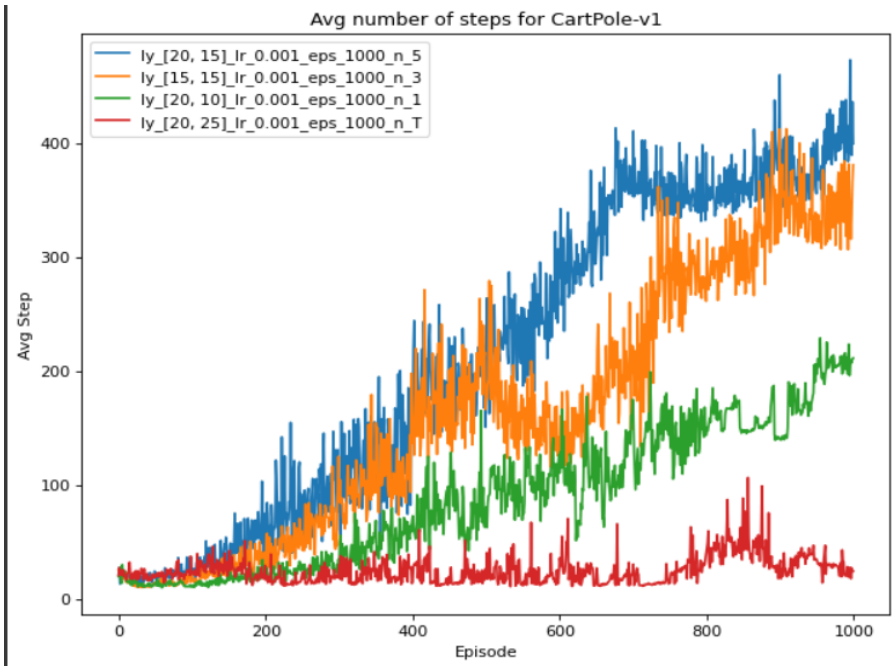
– Avg reward plot:



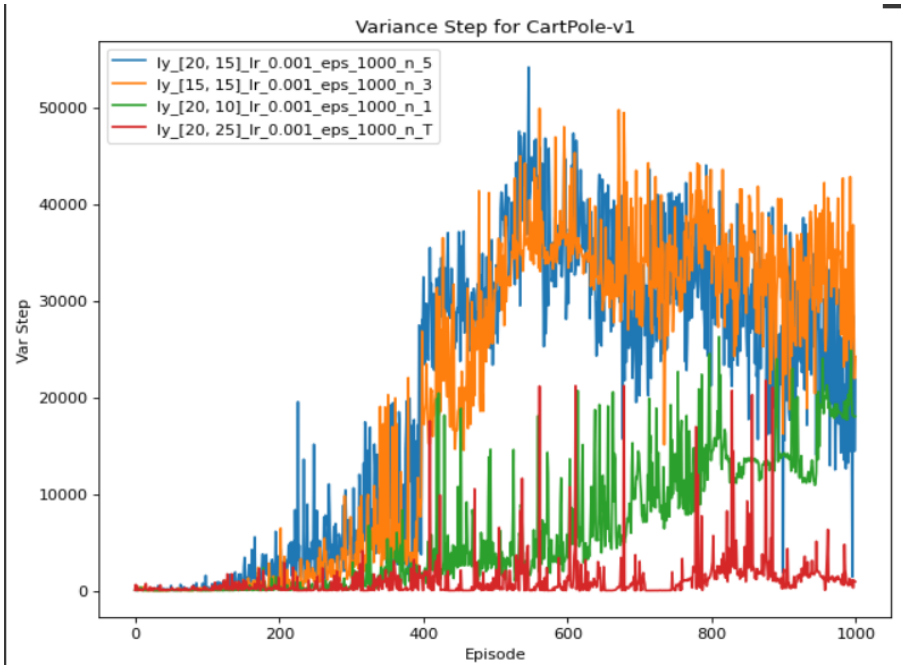
– Variance reward plot:



– Avg step to reach goal plot:



– Variance step plot:



2. **Acrobot-v1**

– Observations:

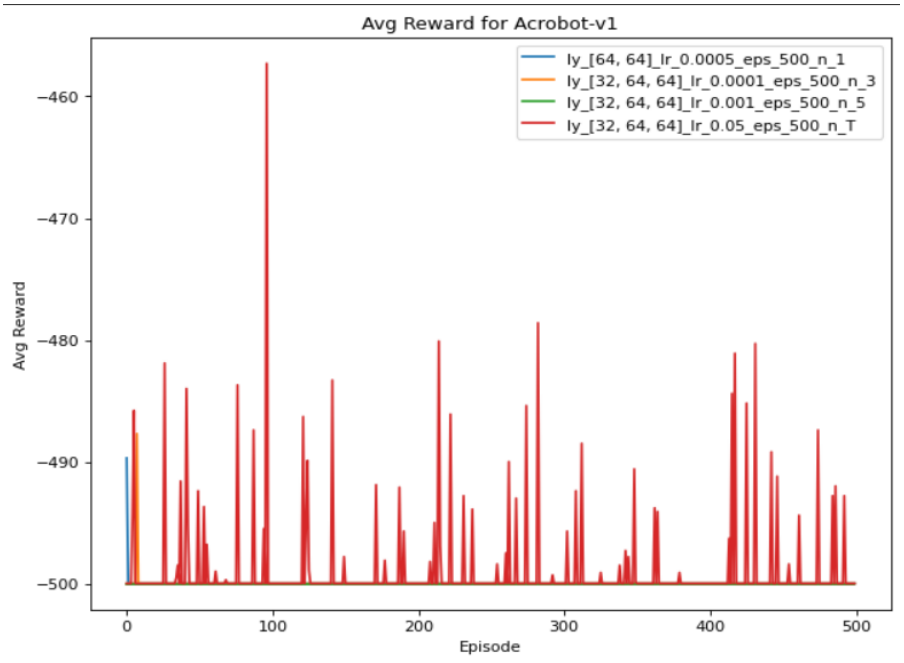
  * Of all the hyper-parameter we experimented except for the following hyper-parameters, almost all gave a constant average reward of -500.

  * Even after averaging with 10 runs we were not able to generate a smooth plot, so that we can compare the performance.

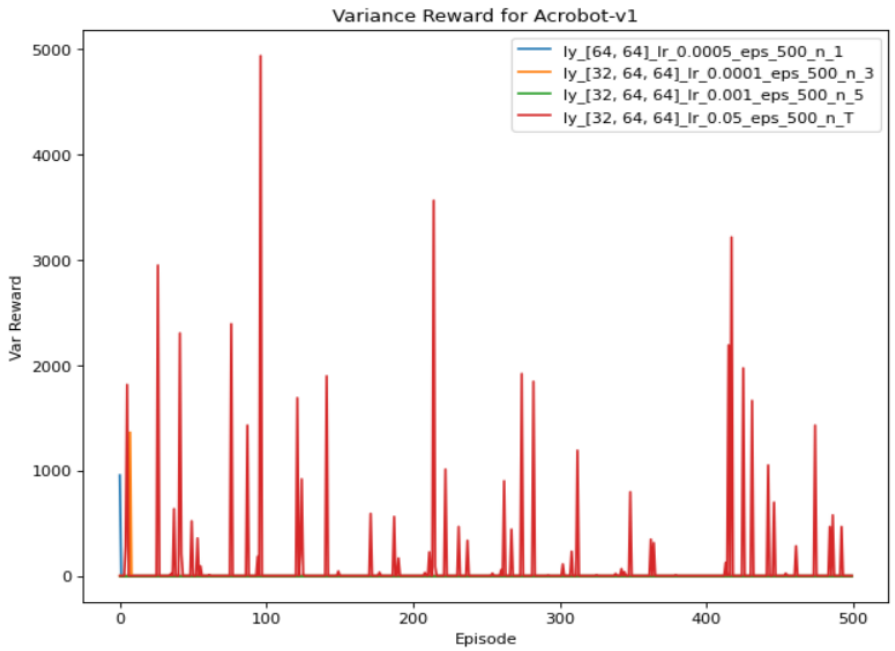
  * We trained our model with only 500 episodes. We believe that if we had better compute resource, we could have trained our model for large number of episodes and these hyper-parameters could have learned good policy.

  * It is difficult to interpret from plot, which AC variation has high variance.

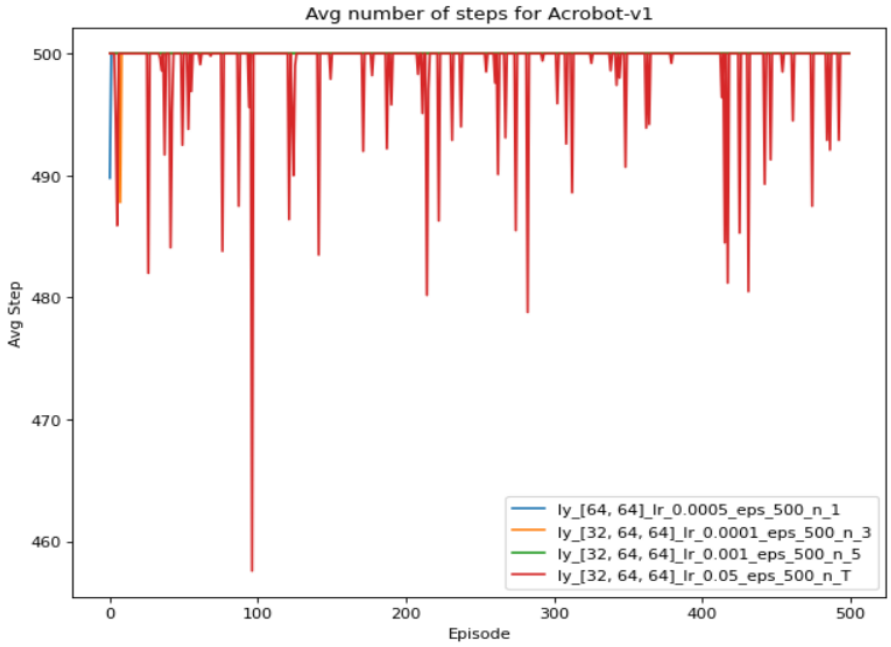
– Avg reward plot:



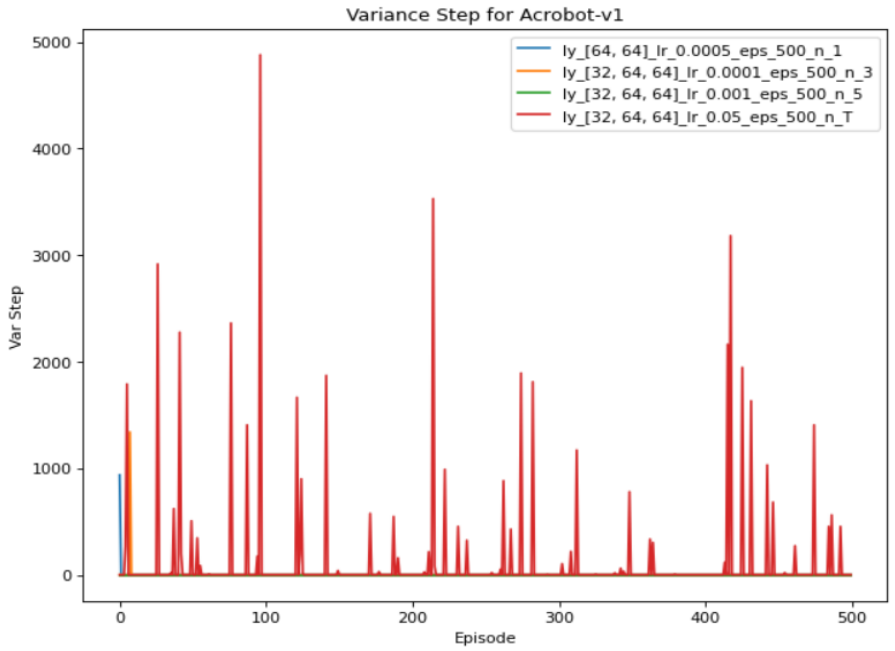
– Variance reward plot:



– Avg step to reach goal plot:



– Variance step plot:

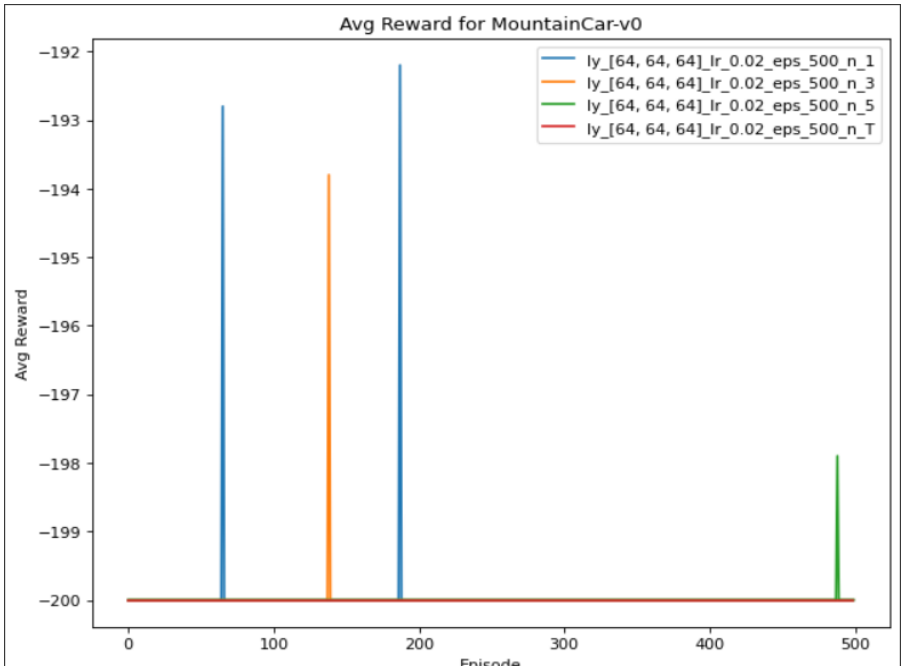


3. **MountainCar-v0**

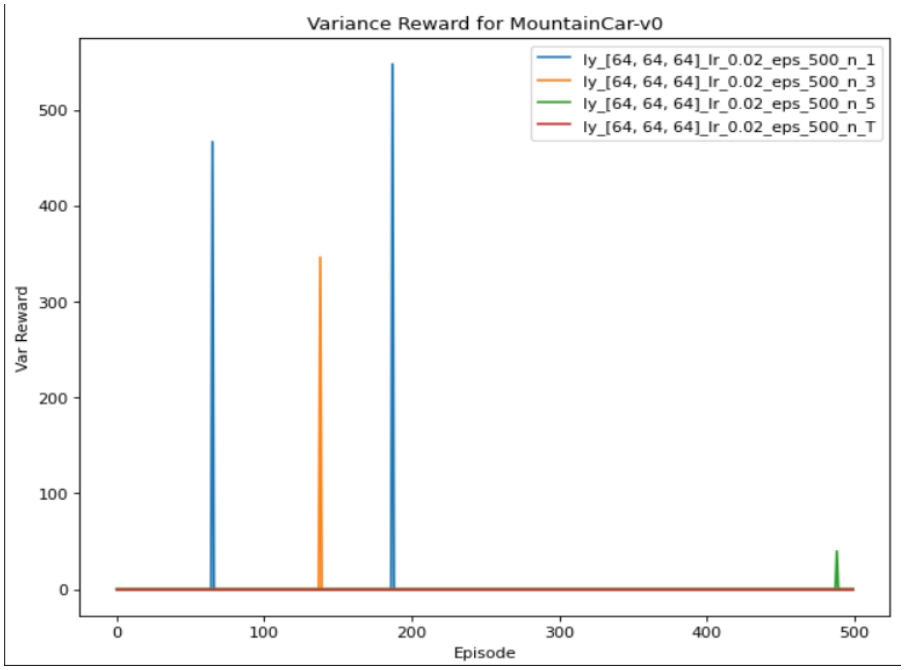
   – Observations:

   * Of all the hyper-parameter we experimented except for the following hyper-parameters, almost all gave a constant average reward of -200.
   * Even after averaging with 10 runs we were not able to generate a smooth plot, so that we can compare the performance.
   * We trained our model with only 500 episodes. We believe that if we had better compute resource, we could have trained our model for large number of episodes and these hyper-parameters could have learned good policy.
   * It is difficult to interpret from plot, which AC variation has high variance.

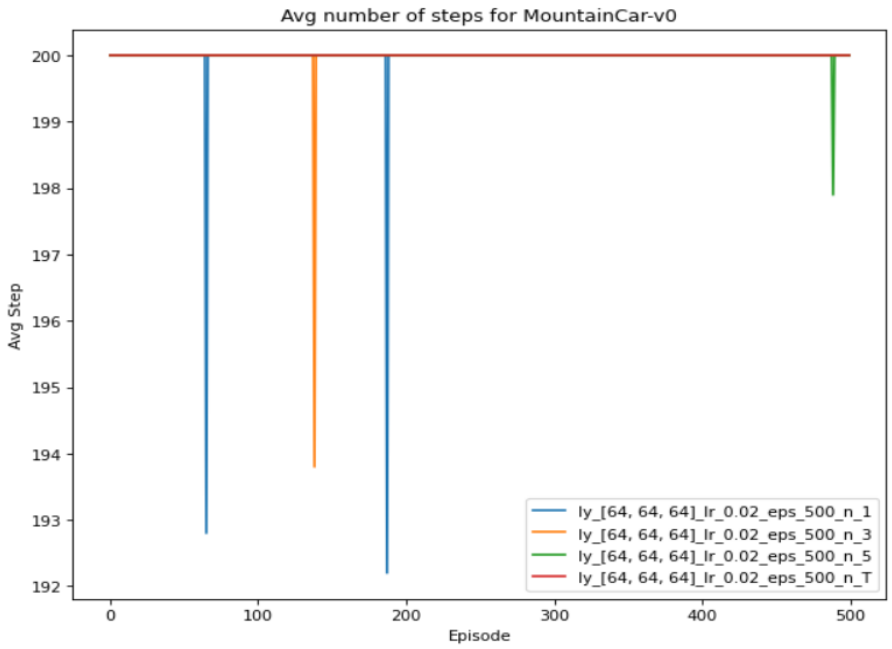
   – Avg reward plot:



– Variance reward plot:



– Avg step to reach goal plot:



– Variance step plot: