CS6700 - Reinforcement Learning
Programming Assignment 2

March 13, 2022

1 Environments

This exercise aims to familiarize you with **DQN and Actor-Critic** methods. You will implement your algorithms on the following four OpenAI Gym environments.

- `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.

2 Tasks

DQN

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

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

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

NOTE: You are graded equally for the experiments, justifications and explanations provided. Some parameters you could play around with are - neural network architecture, learning rate, replay buffer size, batch size, update frequency of target network, truncation limit, discount factor and control parameter.

Actor-Critic

For **Actor-Critic**, experiment with the following aspects (for each environment) to improve the performance, by starting with the network provided in the tutorial:

- Variations of Actor-Critic methods [where $v(s)$ is a parametrized state value function]:
  1. One-step, where $\delta_t^{(1)} = R_{t+1} + \gamma v(s_{t+1}) - v(s_t)$
  2. Full returns, where $\delta_t^{(T)} = \sum_{t'=t}^{T} \gamma^{t'-t} R_{t'+1} - v(s_t)$; T being the maximum number of time steps considered.
  3. n-step returns, where $\delta_t^{(n)} = \sum_{t'=t}^{n+t-1} \gamma^{t'-t} R_{t'+1} + \gamma^n v(s_{n+t}) - v(s_t)$; you can implement for two values of n (chosen appropriately).
- Number and sizes of hidden layers
- Learning rate α
- Number of episodes

You can fix the discount factor $\gamma = 0.99$ for all Actor-Critic experiments. Use the average of at least 10 runs for each variation. Also, note the variances of total episode rewards across the 10 runs for each episode. Similar to DQN, plot reward curves and print the number of steps to reach the goal in each episode for each variation. Observe and comment on the differences in the performance of each of the three AC variations in terms of the number of steps to reach the goal and variance. Which AC method has the highest variance (you can determine this by plotting the variance of the episode reward for each episode)?

For the full returns AC scenario, the actor and critic loss functions are

$$L_{actor} = -\sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \delta_t^{(T)}$$

$$L_{critic} = \sum_{t=1}^{T} \delta_t^{(T)^2}$$

And for the n-step returns scenario, the actor and the critic loss functions are

$$L_{actor}^{(t)} = -\log \pi_\theta(a_t|s_t) \delta_t^{(n)}$$

$$L_{critic}^{(t)} = \delta_t^{(n)^2}$$

where $\delta_t^{(n)} = \sum_{t'=t}^{n+t-1} \gamma^{t'-t} R_{t'+1} + \gamma^n v(s_{n+t}) - v(s_t)$

3 Submission Instructions

You are required to submit both your report and your code on Gradescope. Please submit in **teams of two**. One submission per team will suffice. The due date for this programming assignment is **11:59 pm on Sunday, March 27^{th}**.

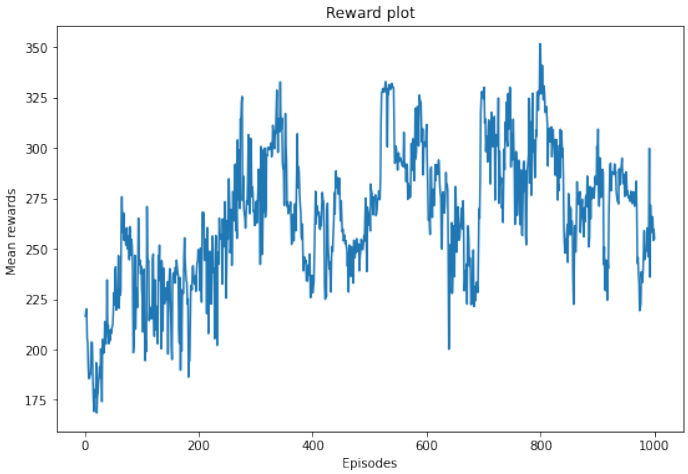
Actor-Critic

CartPole

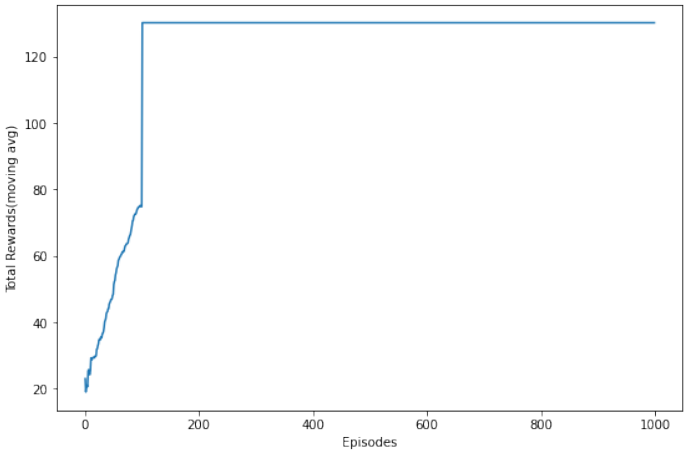
One-step Return

Best hyper-parameters:

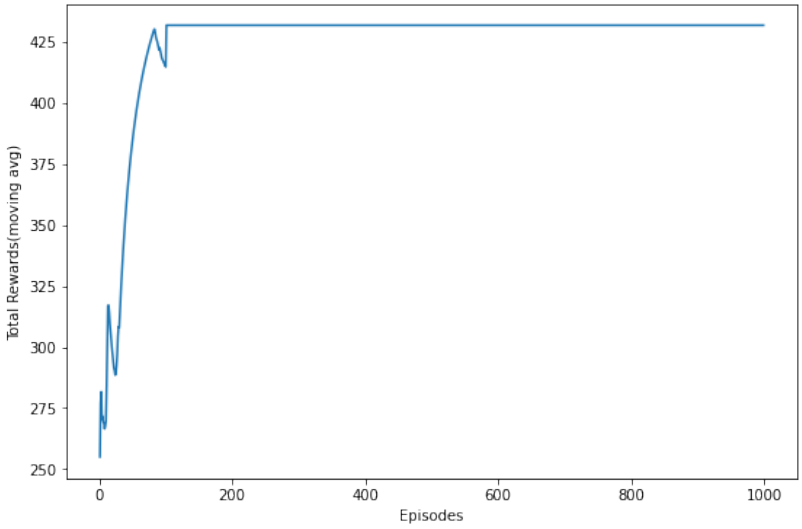
- Single hidden layer with 512 units
- Learning rate = 0.0001



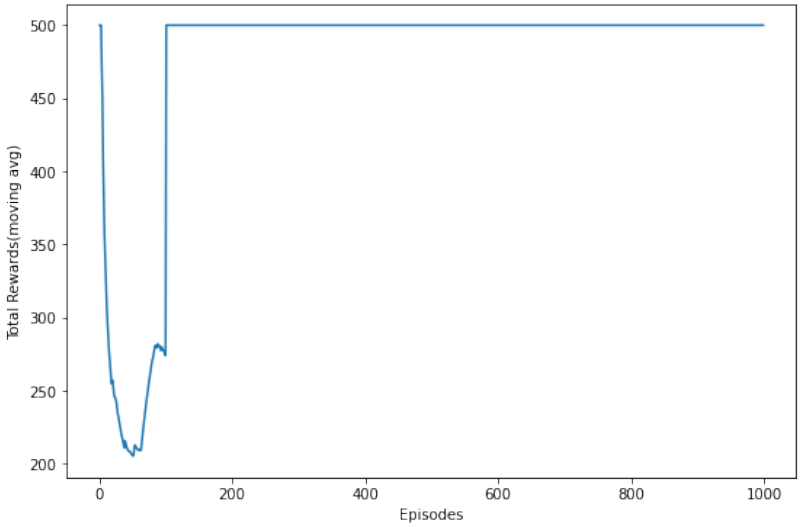
This is the best mean reward plot(for 10 runs) that we could obtain for one-step cartpole. To look further into this we plotted 3 of the runs.



The agent converged to a sub-optimal policy(achieving a little more than 120 steps).

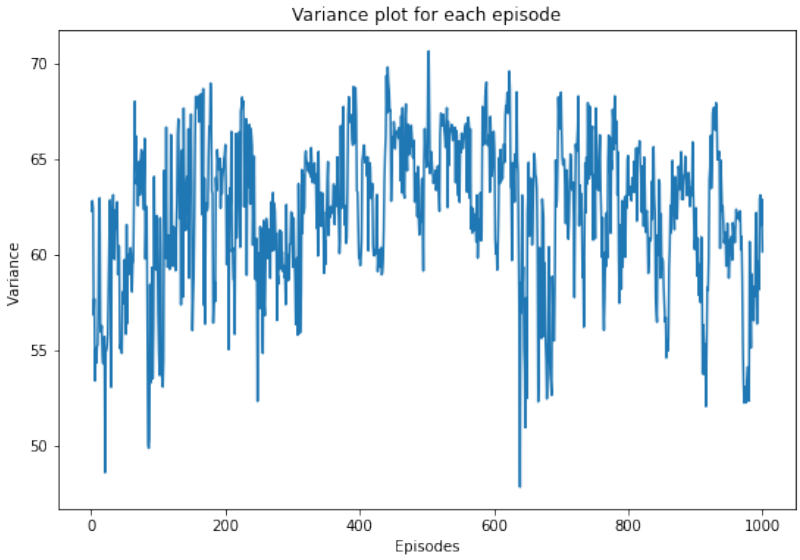


Here we have a much better policy but it still doesn't achieve the threshold of 475.



In the above case, we believe that the initial seed somehow helped it go past the threshold to a reward of 500, and also in two other runs too, the agent starting with a very high initial reward was able to achieve the threshold.

So even for tuned hyper-parameters, one-step cartpole may not be very optimal when compared to full return or n-step return.

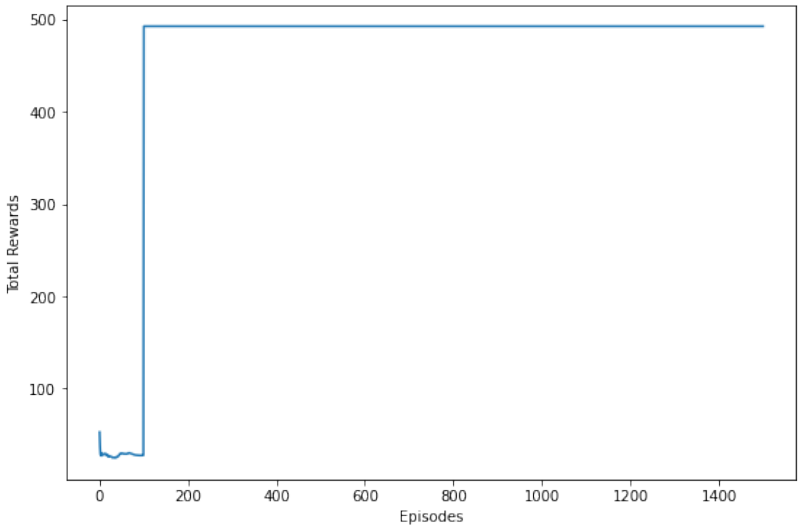


The variance for most of the episodes is greater than 60.

Full Return

Best hyper-parameters:

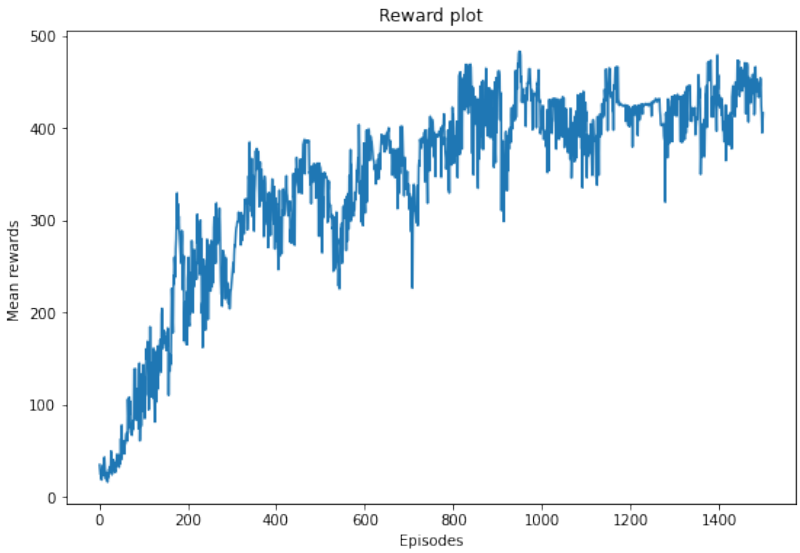
- Single hidden layer with 128 units
- Learning rate = 0.001



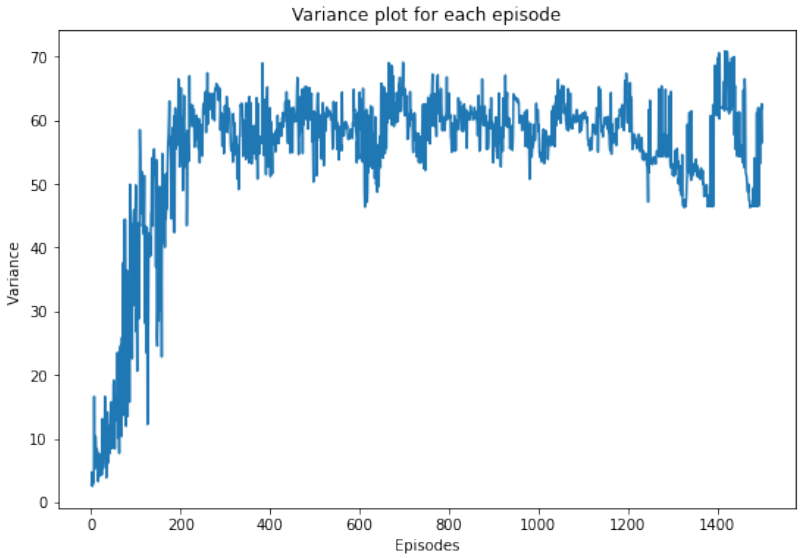
The above curve is for a single run(reward averaged over previous 100 episodes) wherein the

agent learns an optimal policy i.e. the cartpole is able to sustain the pole for nearly 500 steps.

In case of cartpole, the number of steps to reach the goal is equivalent to reward itself. Lets plot the reward curve for 10 runs



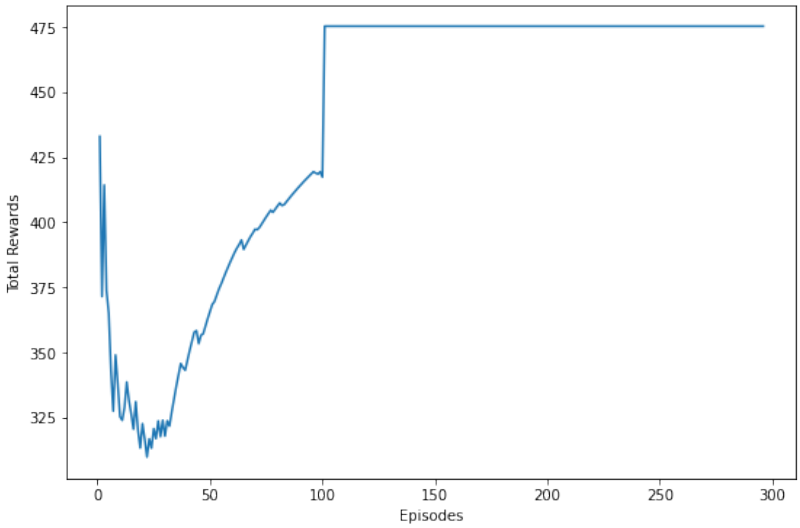
Out of the 10 runs, 2-3 of them perform poorly(don't reach threshold), due to which we observe the above aberrations in the mean reward curve.



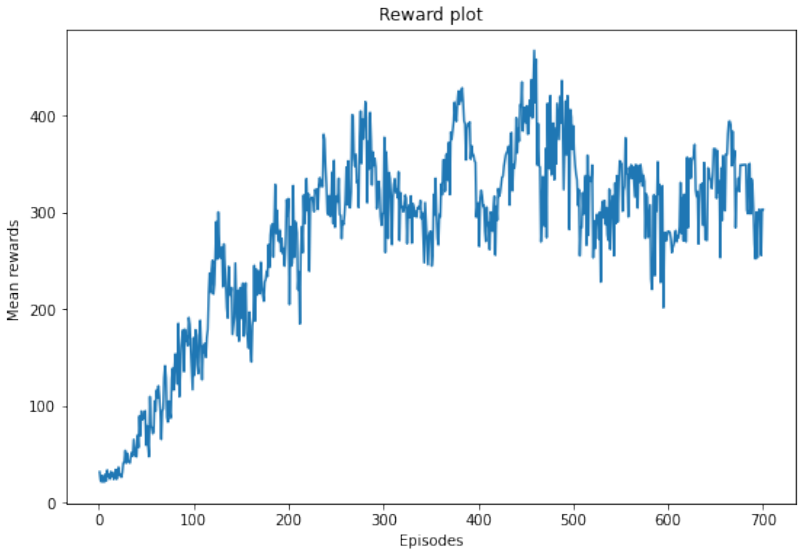
n-step Returns

Best hyper-parameters:

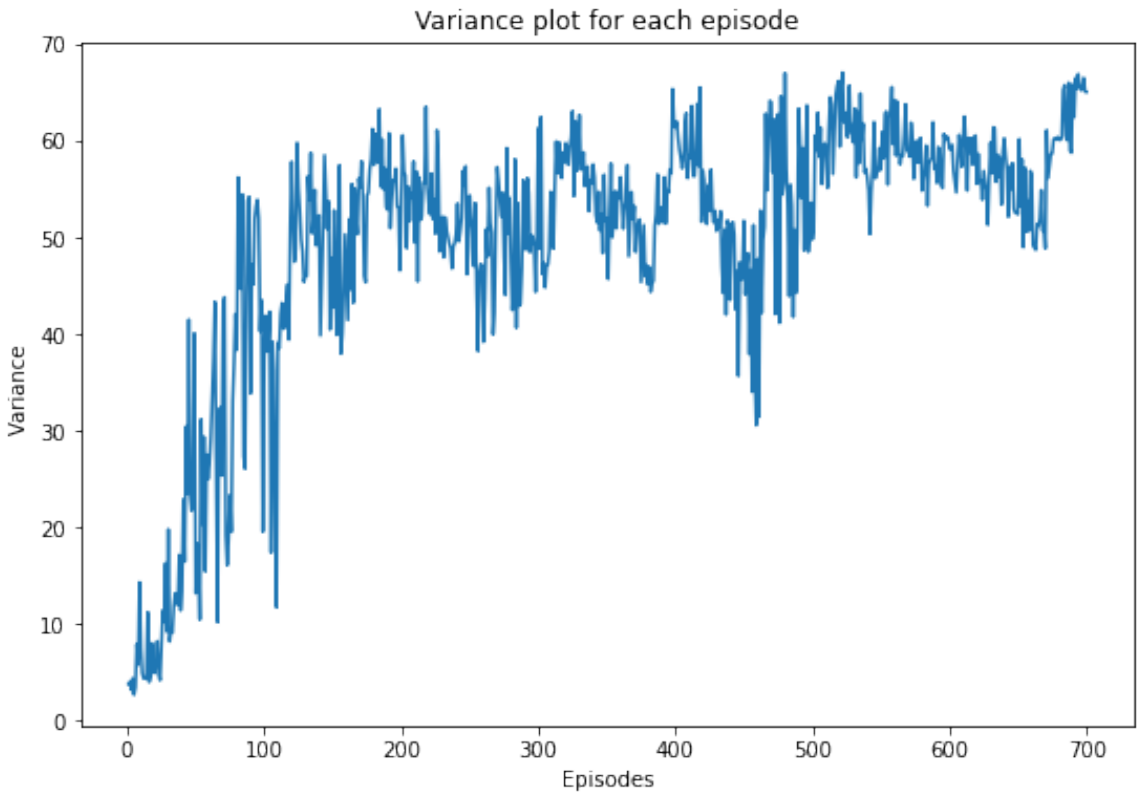
- Single hidden layer with 128 units
- Learning rate = 0.001
- n=10



The above curve is for a single run(reward averaged over previous 100 episodes), in 300 episodes it reaches the threshold of 475.



Mean reward curve
Computed it till 700 episodes beyond which it doesn't improve significantly.



Among the three the variance is higher for one-step and full return when compared to n-step. High variance for one-step is observed owing to the dependence of returns on the initial seed. Some of the initial seeds may allow it to have very high rewards and the other times low rewards.

Full return seems to perform the best for cartpole in terms of its mean reward obtained.

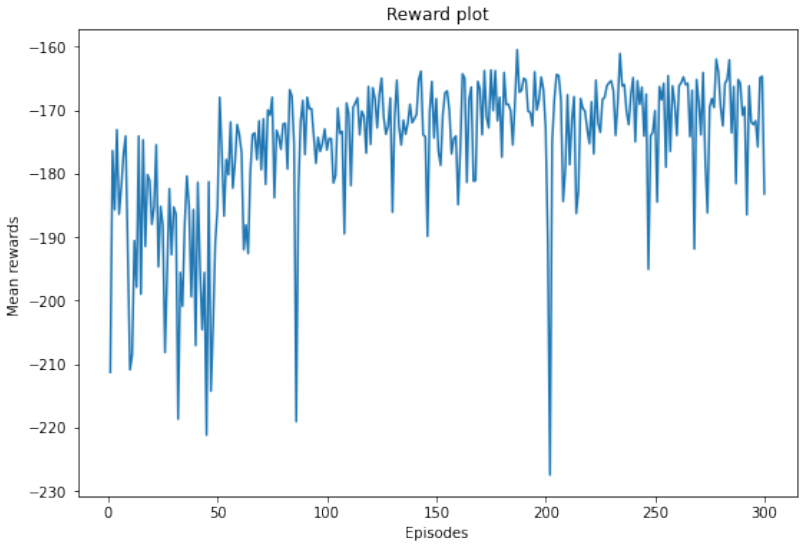
Acrobot

One-step Return

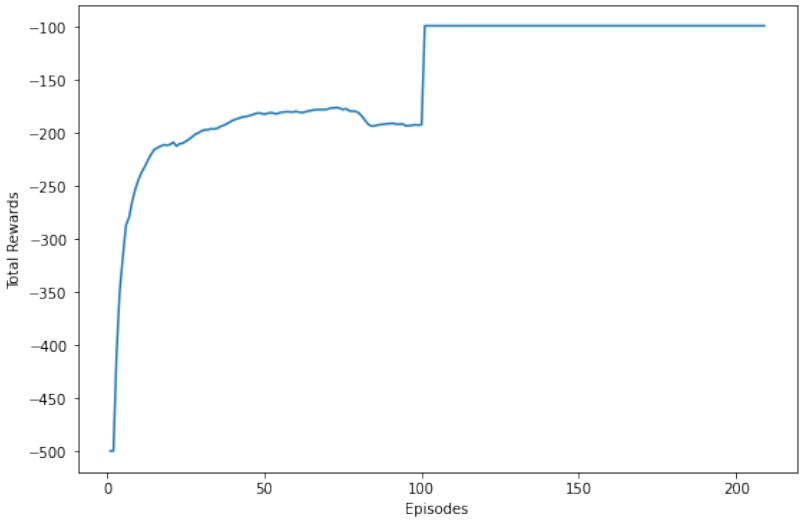
Best hyper-parameters:

- Two hidden layer with sizes 512 and 512
- Learning rate = 0.0001

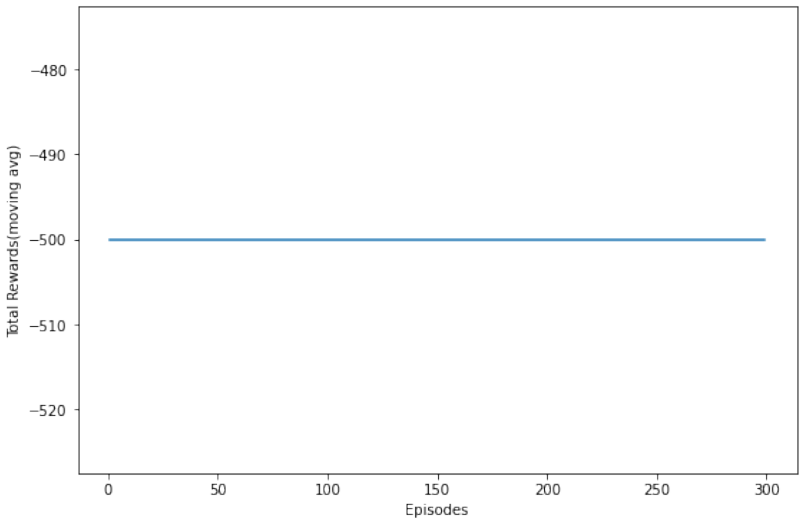
One important hyper-parameter change that helped us in Acrobot is the additional layer to the neural network. This makes sense as we expect the value function, policy of Acrobot to be much more complex than cartpole

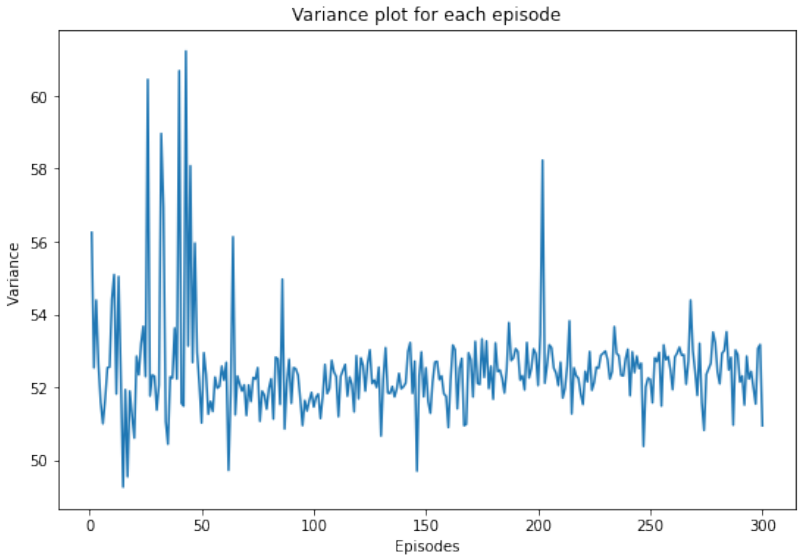


Mean reward plot over 10 runs. Its much lower than -100. Lets again take a peek into the individual runs.



The above one is the optimal run we would expect.
Among the 10 runs we also have 2 runs in which the agent is simply stuck at -500 reward. Because of these particular runs we expect the variance to increase substantially.



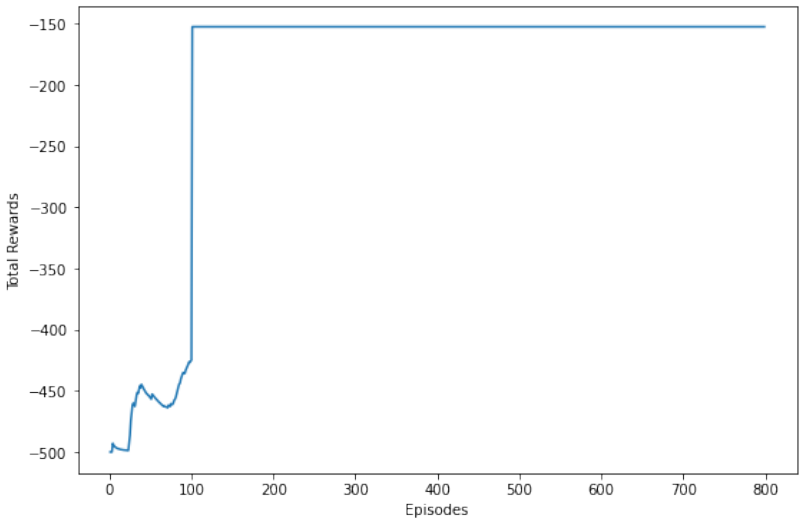


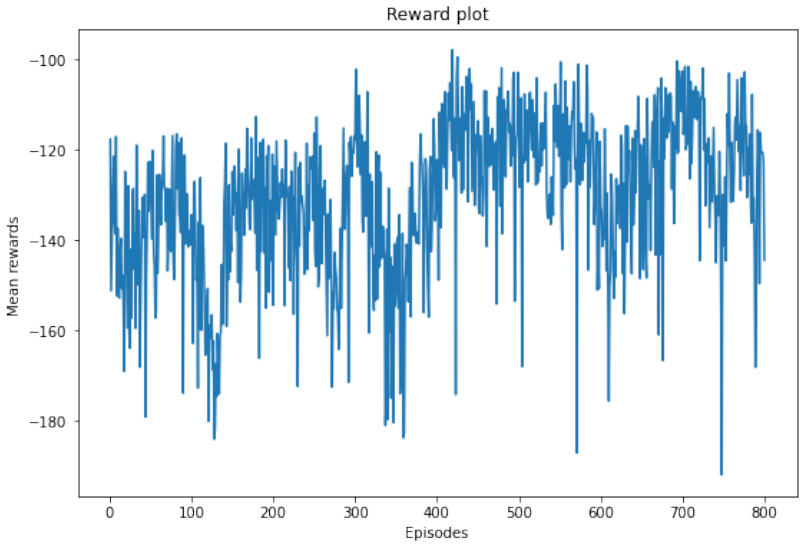
Observe substantially higher variance.

Full Returns

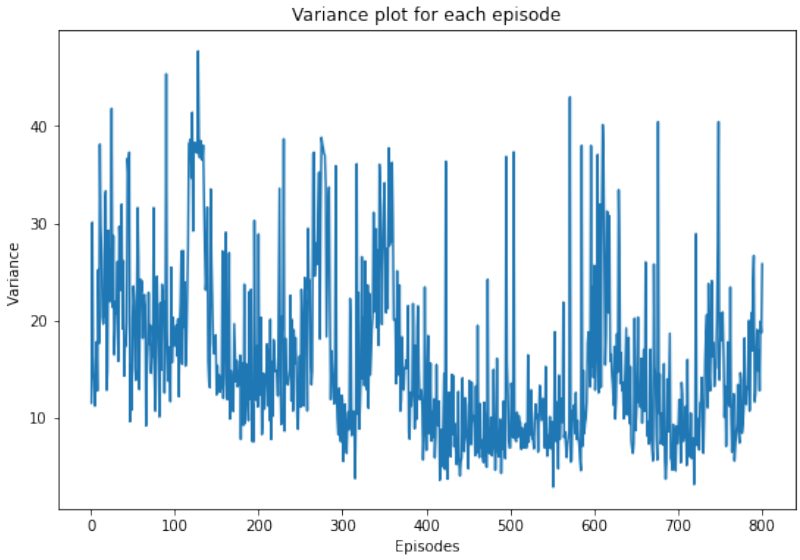
Best hyper-parameters:

- Two hidden layer with sizes 512 and 1024
- Learning rate = 0.001





Converges close to (-140) - (-130).

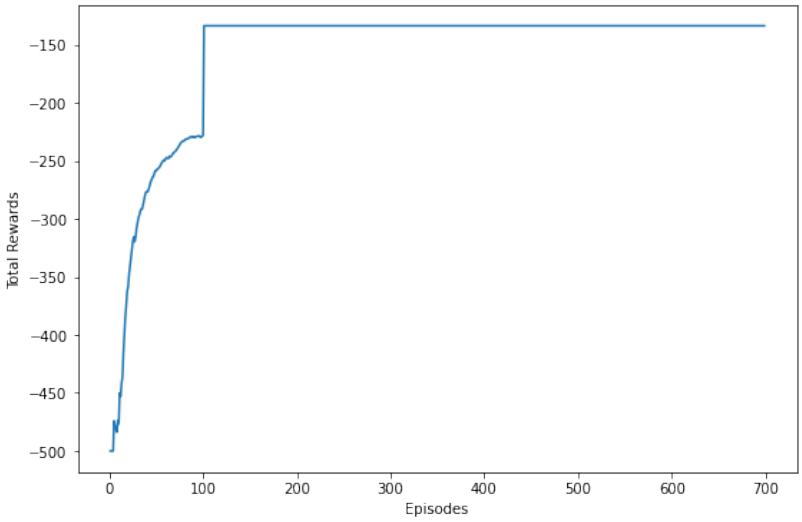


Has considerably low variance

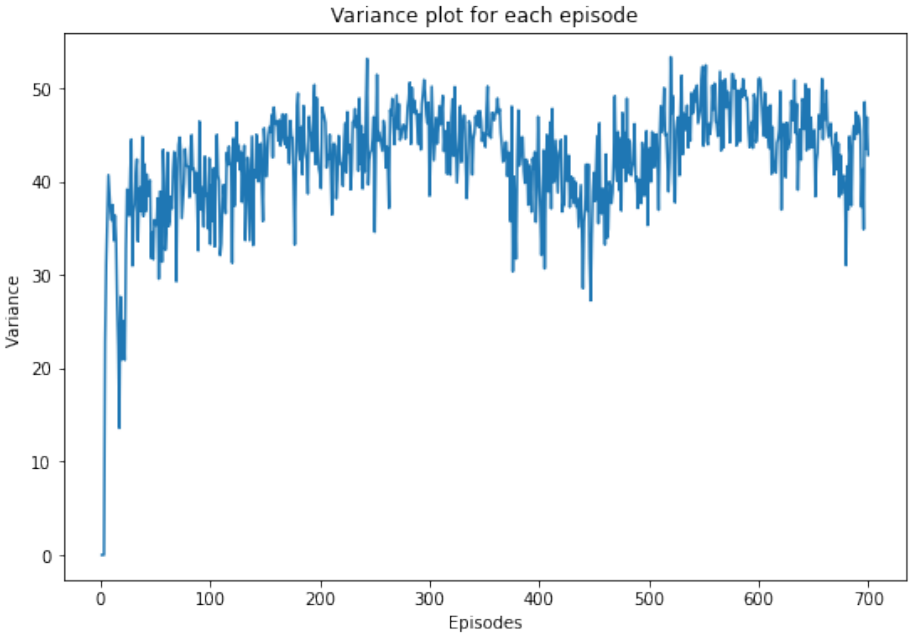
n-step Returns

Best hyper-parameters:

- Two hidden layer with sizes 512 and 1024
- Learning rate = 0.001



This converges close to -140

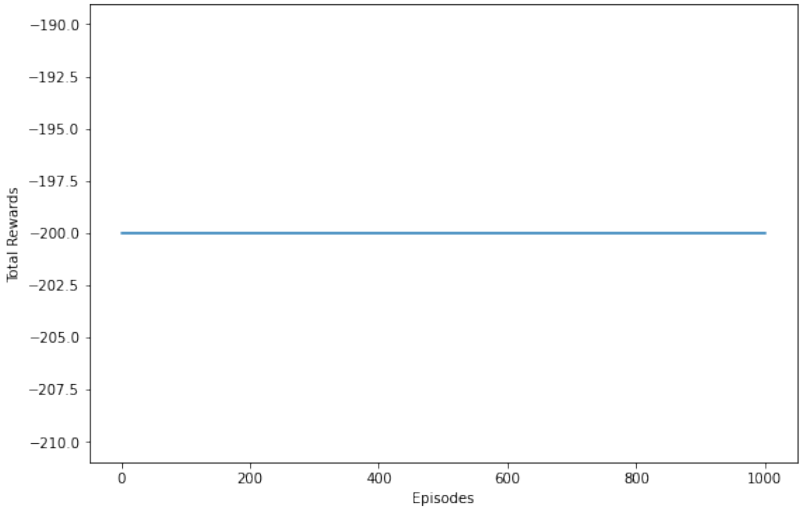


In case of acrobot, one-step return has the highest variance again owing to the 2 outlier runs which are stuck at -500 reward. Also note that all the other 8 runs perform the best i.e. they achieve rewards greater than the threshold of -100. Compare it with full return which achieves a median reward close to -130 but has lower variance. n-step is intermediate between the two in terms of its variance.

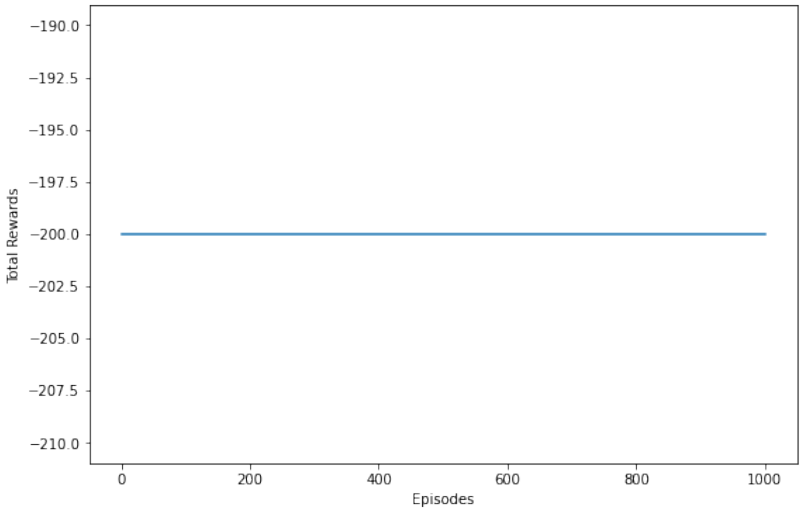
Mountain-car

We've tried for sufficient number of hyper-parameters, but couldn't quite get it to go past reward of -200. We believe that in actor-critic there is not as much scope for exploration as in DQN, which is causing these results.

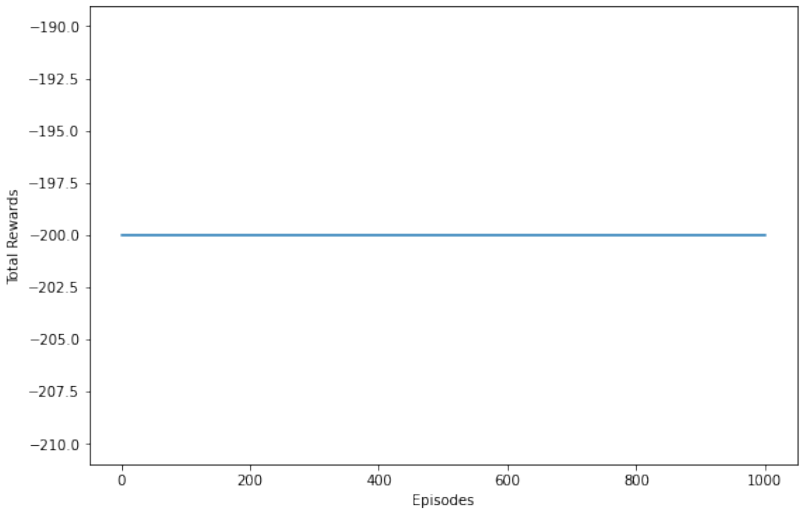
One-step Return



full Returns



n-step Returns



Cartpole-v0

Detailed work of all our Exerpimets can be found in this Dashboard Wandb Dash-Board for RLPA2

1 Environment and Criterion

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.

The convergence criterion is if the average score across 100 episodes is above 195.

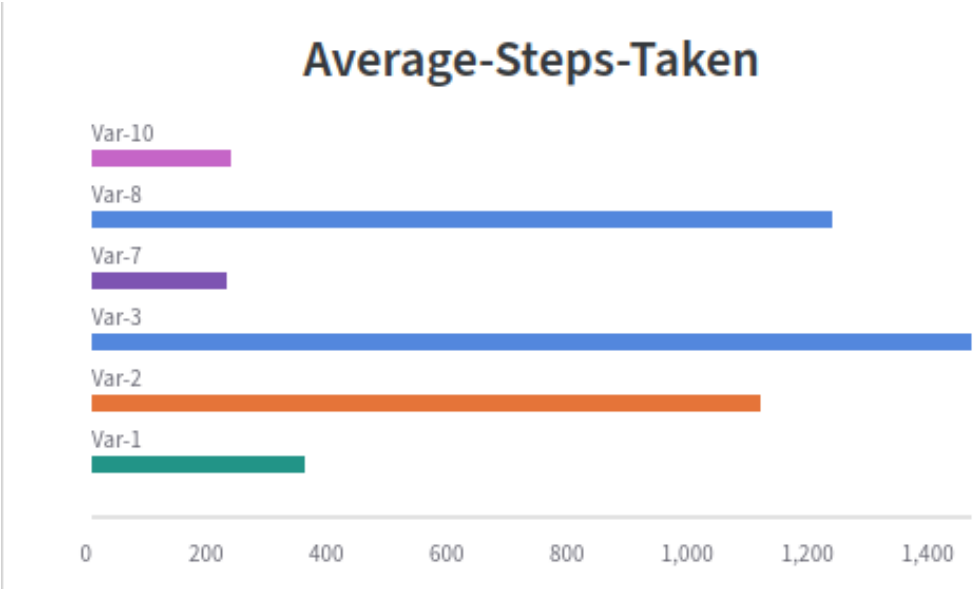
2 Introduction

We conducted 10 experiments and found 5 model parameters that increase successively. The parameter experiments are summarized in the tables given below. We give the rationale for parameter tuning in the following sections.

The five successive models are

$$7 > 10 > 1 > 2 > 8 > 3$$

Figure 1: Average Steps taken by Best 6 Hyper Parameter Combinations



Parameters	1	2	3	4	5
BUFFER_SIZE	int(1e5)	int(1e5)	int(1e5)	int(1e3)	int(1e4)
BATCH_SIZE	64	64	64	64	64
GAMMA	0.99	0.99	0.99	0.99	0.99
LR	5.00E-04	5.00E-04	5.00E-04	5.00E-04	5.00E-04
UPDATE_EVERY	20	5	30	20	20
POLICY	eps	eps	eps	eps	eps
Avg steps taken	353.6	1111.5	1463	786.1	5402(2 runs)

Parameters	6	7	8	9	10
BUFFER_SIZE	int(1e6)	int(1e5)	int(1e5)	int(1e5)	int(1e6)
BATCH_SIZE	64	128	128	128	128
GAMMA	0.9	0.999	0.999	0.999	0.999
LR	5.00E-04	1.00E-04	5.00E-03	5.00E-05	1.00E-04
UPDATE_EVERY	20	20	20	20	30
POLICY	eps	eps	eps	eps	eps
Avg steps taken	6900	224	1233	2564 (2 runs)	233

3 Tuning Update_every (Exp1, 2, 3)

For the first three experiments we varied the Update_every parameter between 20, 5, 30. From the above results Table it can be inferred that 20 works best. There is a vast difference between the results. The model seems quite sensitive to the parameter.

Too high or too low episodes for updating the target network is deteriorating the performance. This could be because, too high a parameter would imply that the target network is adapting very slowly. Too low a parameter defeats the purpose of target network being updating separately. This makes the model more unstable. We will be using mostly 20 in the further experiments

Plots of Exp1 , Exp2, Exp 3

Figure 2: Number of steps taken by Experiment 1 for 10 times

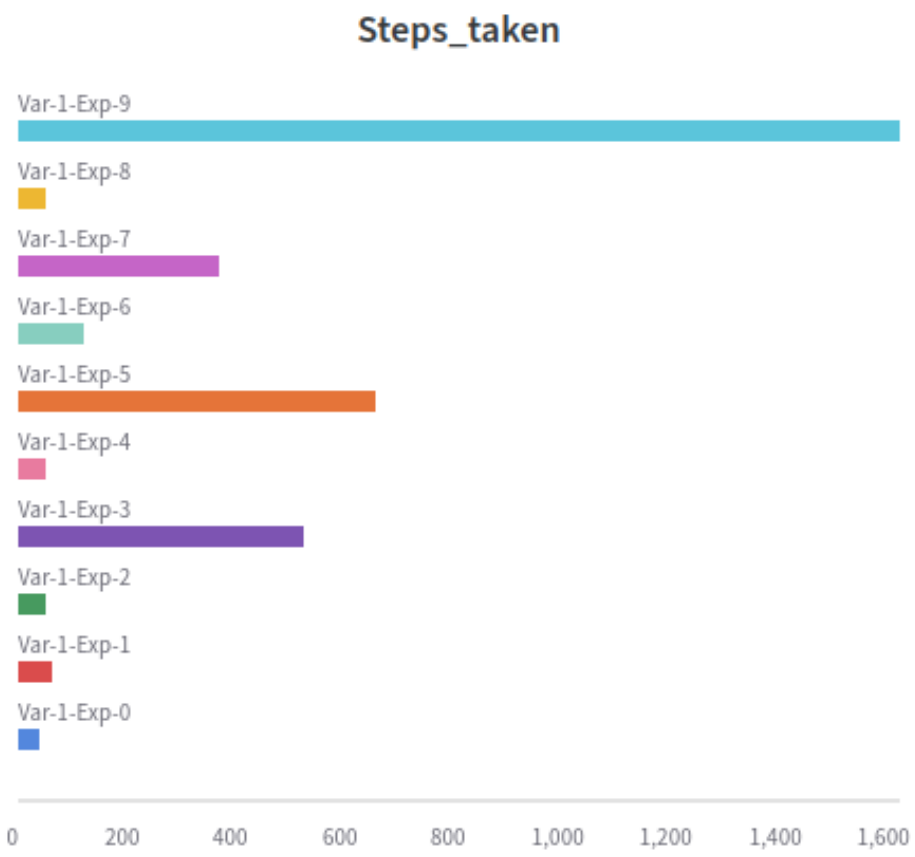


Figure 3: Number of steps taken by Experiment 2 for 10 times

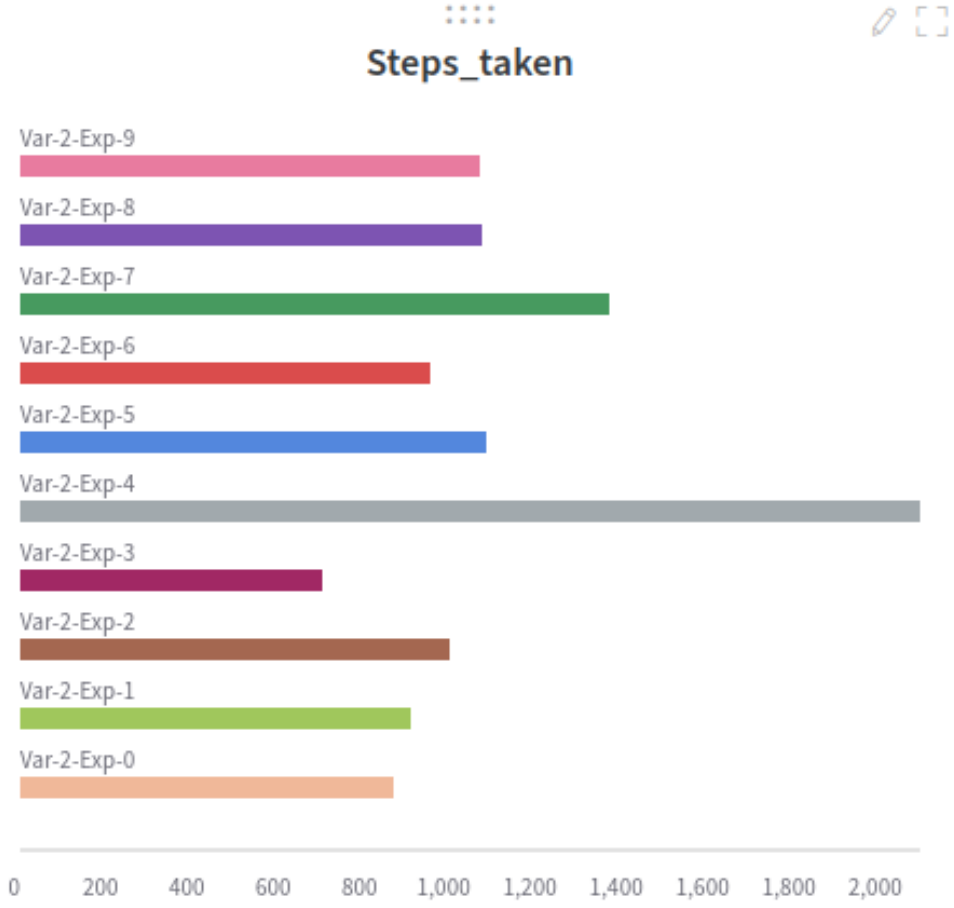


Figure 4: Number of steps taken by Experiment 3 for 10 times

4 Tuning Buffer size(Exp1, Exp4 , Exp5, Exp6)

We turn to tuning Buffer Size, we test four buffer sizes 1e5, 1e3, 1e4, 1e6. From the table it is clear that 1e5 gives the best performance compared to other values. While the reason for this not completely clear. Having more buffer size is computationally more expensive in terms of the storage.

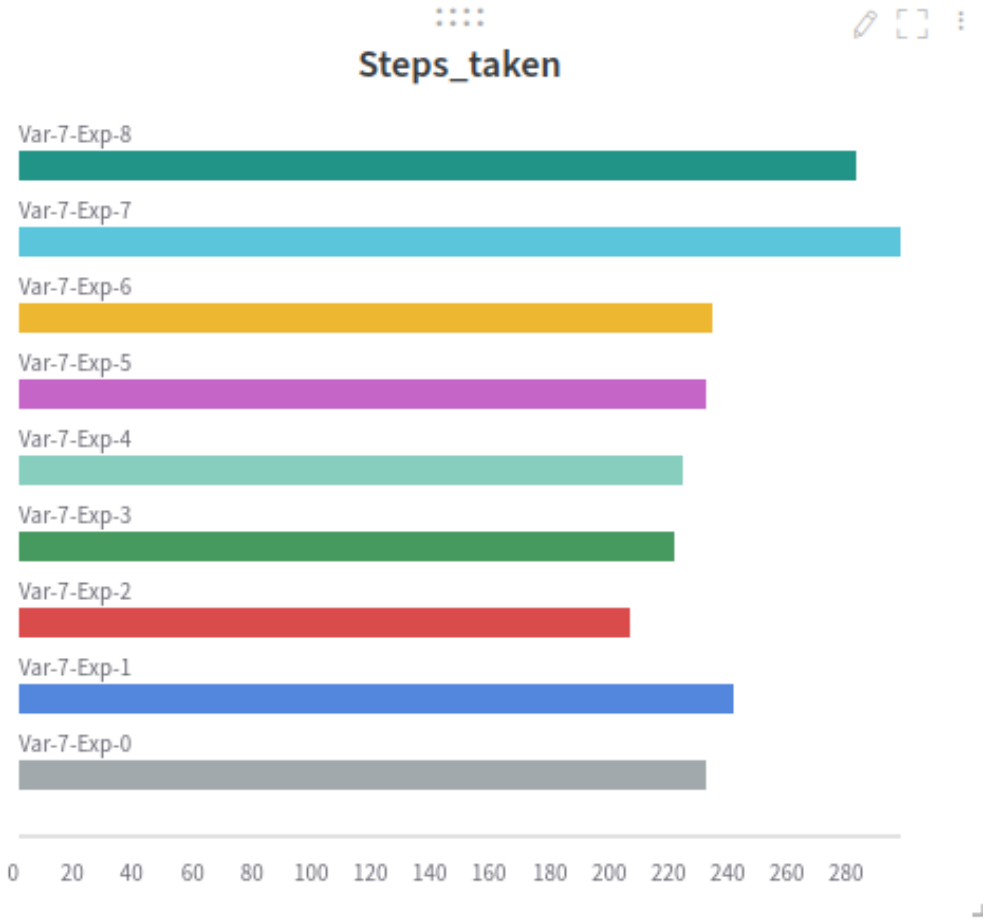
5 Tuning Learning Rate & batch size (Exp 7, Exp 8, Exp 9)

Tuning Learning Rate has shown significant improvement. We tried four learning rates $5.00e-04$ (exp1) , $1.00e-04$ (exp 7), $5.00e-03$ (exp 8), $5.00e-05$ (exp 9

) . Exp 7 has given the best results overall. Exp7 is also very consistent across runs.

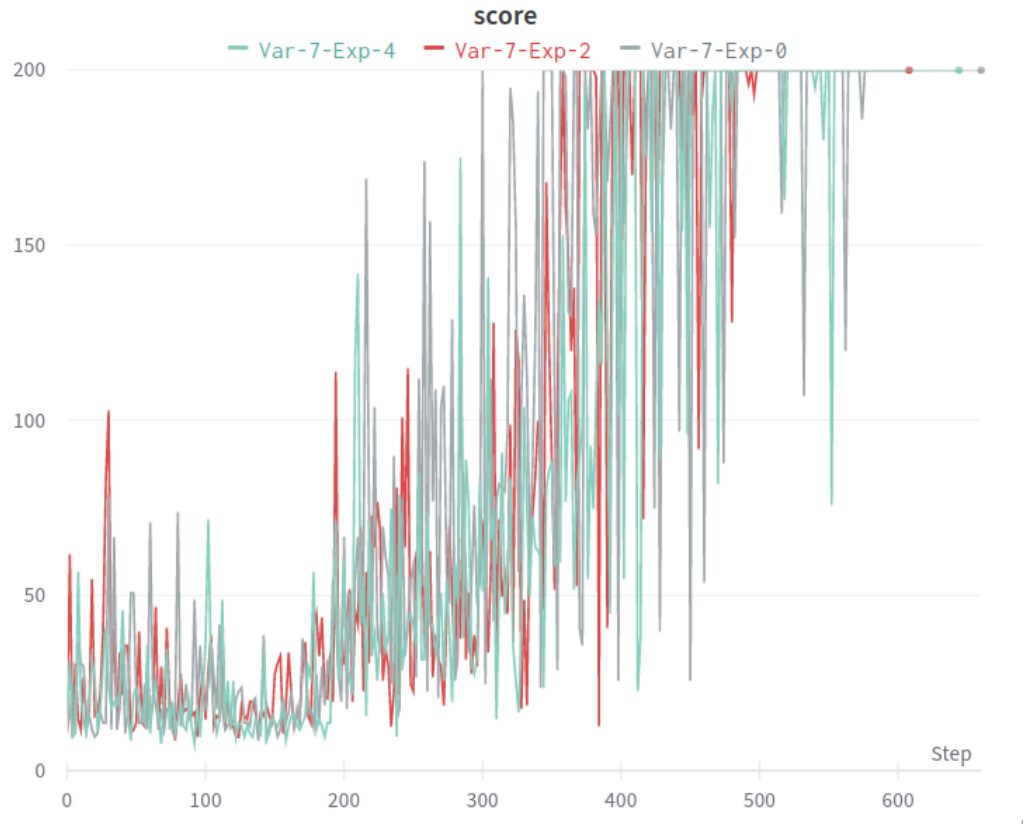
5.1 Plots Exp7

Figure 5: Number of steps taken by Experiment 7 for 10 times



It can be seen that all the runs converged below 300 episodes.

Figure 6: Reward Curves for few of the runs for Experiment 7



6 Best Parameters

From the Experiments we conclude that Variation 7 was the best, here are the corresponding Hyper Parameters

Parameters	**7**
BUFFER_SIZE	int(1e5)
BATCH_SIZE	128
GAMMA	0.999
LR	1.00E-04
UPDATE_EVERY	20
POLICY	eps
Avg steps taken	224

Arcobat

Detailed work of all our Exerpimets can be found in this Dashboard Wandb DashBoard for RLPA2

1 Introduction

In this section we train the agent called Arcobat-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.

1.1 Converging Criterion

After multiple experiments we settled for converging criterion of Average for 100 episodes is greater than -80.

2 Best 5 Variations

Parameters	1	2	4	6	7
BUFFER_SIZE	int(1e5)	int(1e5)	int(1e5)	int(1e5)	int(1e5)
BATCH_SIZE	64	64	128	512	512
GAMMA	0.99	0.99	0.99	0.999	0.99
LR	5.00E-04	5.00E-04	1.00E-04	1.00E-05	5.00E-03
UPDATE_EVERY	20	20	20	20	20
POLICY	eps	eps	eps	eps	eps
LAYERS	3	4	3	3	3
Avg steps taken	764	796.2	1127	1900	919.5

The five successive models are

$$1 > 2 > 7 > 4 > 6$$

The below table shows the parameters of all experiments. In the subsequent sections we explain the rational for the variations we followed.

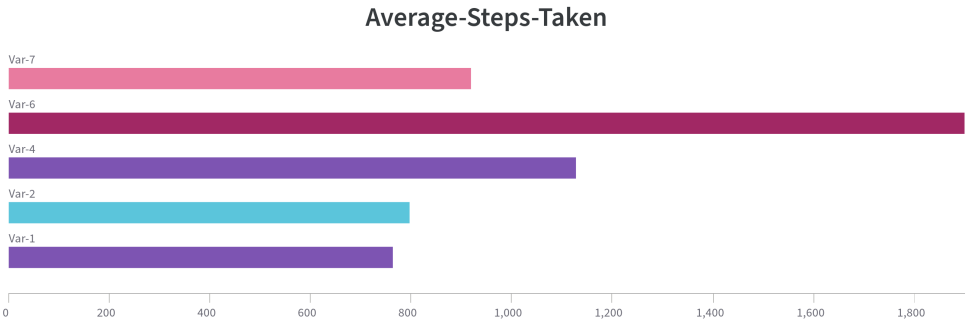


Figure 1: Best 5 Variations Average Scores

3 Variation 4

The parameters for Experiment 4 are

Parameters	4
BUFFER_SIZE	int(1e5)
BATCH_SIZE	128
GAMMA	0.99
LR	1.00E-04
UPDATE_EVERY	20
POLICY	eps
LAYERS	3
Avg steps taken	1127

In this experiment we have varied the learning rate, this is the learning rate is cut by 5th compared to Experiments 1, 2. High Learning rate doesn't seem to help the agent. Experiment 4 is very similar to Experiment, but it can be seen that there is an outlier run which ran for over 2200 episodes, before converging. Even without the outlier the 3rd run is still 1200 episodes.

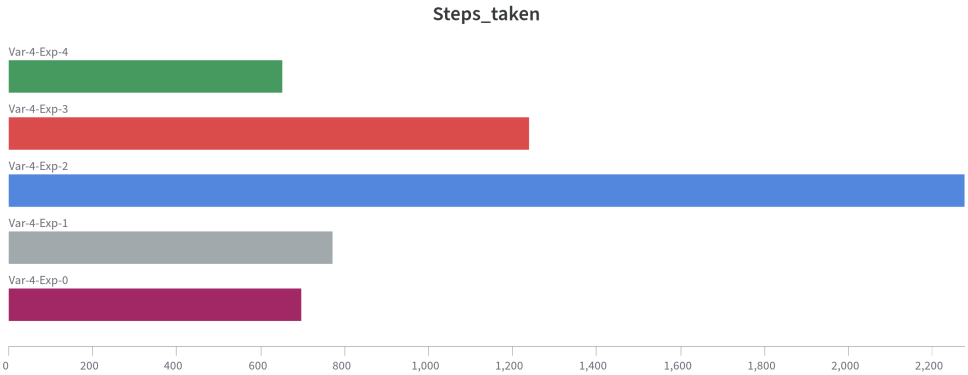


Figure 2: Scores for the best runs with Variation 4

4 Variation 1

The parameters for Experiment 1 are

Parameters	1
BUFFER_SIZE	int(1e5)
BATCH_SIZE	64
GAMMA	0.99
LR	5.00E-04
UPDATE_EVERY	20
POLICY	eps
LAYERS	3
Avg steps taken	764

Variation 1 gave the best Hyperparameters of all the Variations.

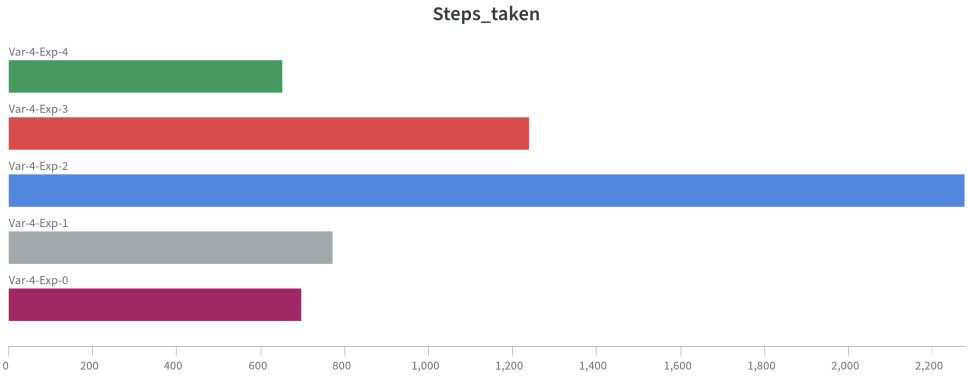


Figure 3: Scores for the best runs with Variation 1

5 Variation 2

The parameters for Experiment 2 are

Parameters	2
BUFFER_SIZE	int(1e5)
BATCH_SIZE	64
GAMMA	0.99
LR	5.00E-04
UPDATE_EVERY	20
POLICY	eps
LAYERS	4
Avg steps taken	796.2

In this Variation we varied the number of layers from Experiment 1. We added an additional Hidden Layer. The first Hidden layer has 256 Nodes, second hidden layer has 128 Nodes, the third hidden layer has 64 Nodes. The results are very similar to Experiment1

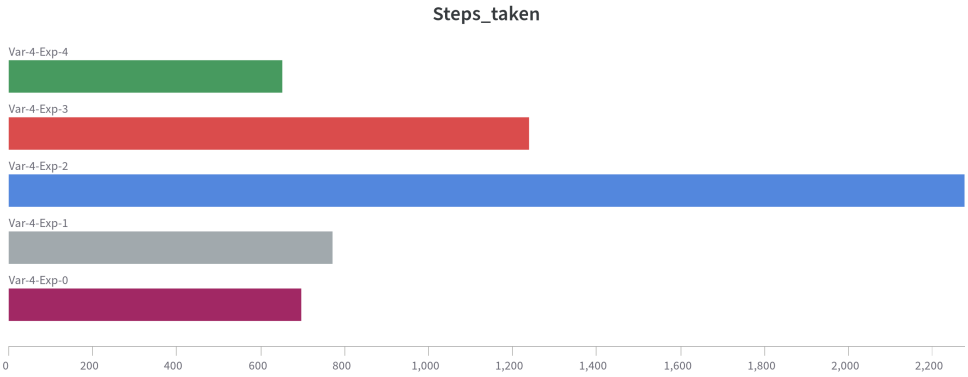


Figure 4: Scores for the best runs with Variation 2

6 Variation 6

The parameters for Experiment 6 are

Parameters	6
BUFFER_SIZE	int(1e5)
BATCH_SIZE	512
GAMMA	0.999
LR	1.00E-05
UPDATE_EVERY	20
POLICY	eps
LAYERS	3
Avg steps taken	1900

Buffer Size = int(1e5) In this Experiment we increased the Batch size, decreased the learning rate. The performance clearly decayed. It took 1900 episodes on average

7 Variation 7

The parameters for Experiment 7 are

Parameters	7
BUFFER_SIZE	int(1e5)
BATCH_SIZE	512
GAMMA	0.99
LR	5.00E-03
UPDATE_EVERY	20
POLICY	eps
LAYERS	3
Avg steps taken	919.5

In this Experiment we increase the Learning 100x. The performance was not very bad.

This is the reward curve for one the runs with this variation

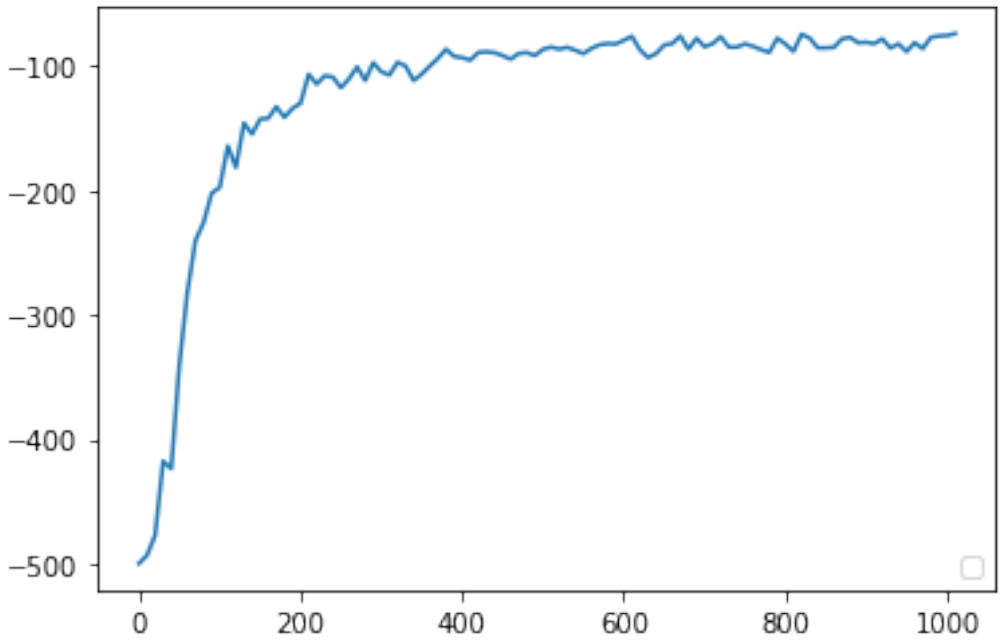


Figure 5: Reward curve for one of the runs with Variation7

Mountain Car

Detailed work of all our Exerpimets can be found in this Dashboard Wandb DashBoard for RLPA2

1 Environment

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.

2 Approch

We tried a lot of Expriments but none of them converged. We give a brief account of all the experiments that failed, until one of them started to show convergence. Then we improved on that.

3 Convergence Criterion

The agent is said to be converged if average score for 100 episodes is greather then -110.0

4 Random Sweep

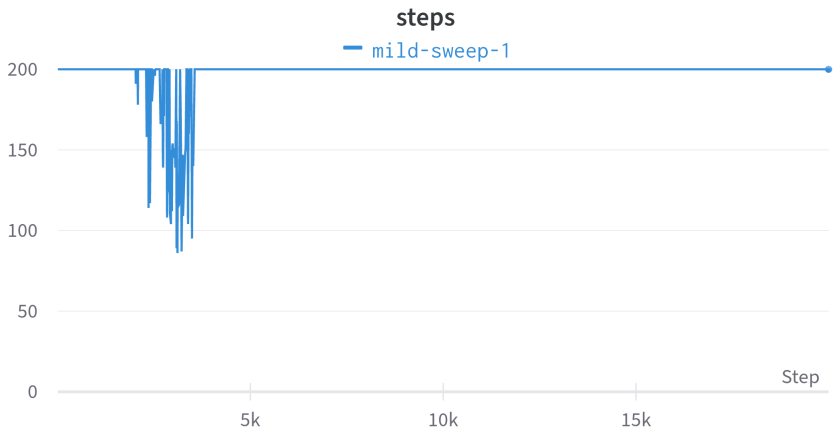


Figure 1:

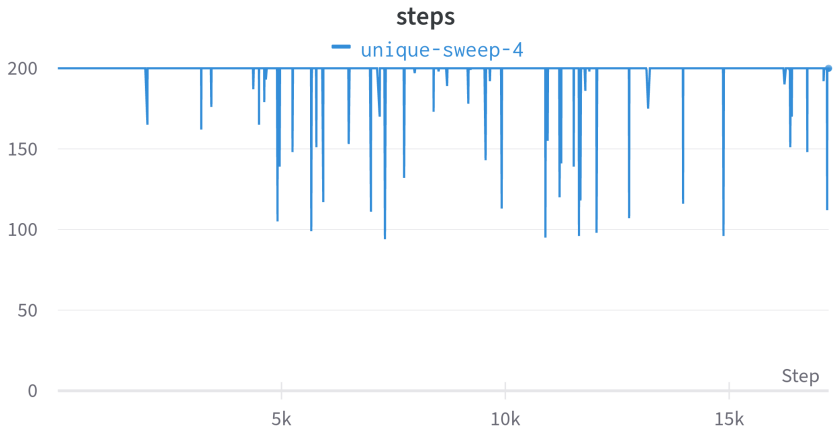


Figure 2:

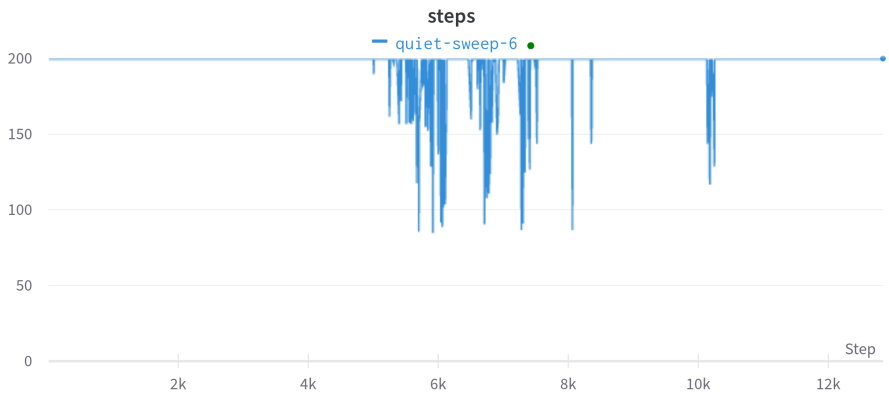


Figure 3:

These are only bad examples, there were lot more runs and sweeps

5 A positive sweep

Figure 4:

The parametrs for the above sweep are the following (put here)

6 Improvement

We took the above parameters and increased the number of layers to 7. (put the layers information here) (put the pic of the convergence)

Layer	Nodes
fc1	128
fc2	256
fc3	512
fc4	1024
fc5	128
fc6	64

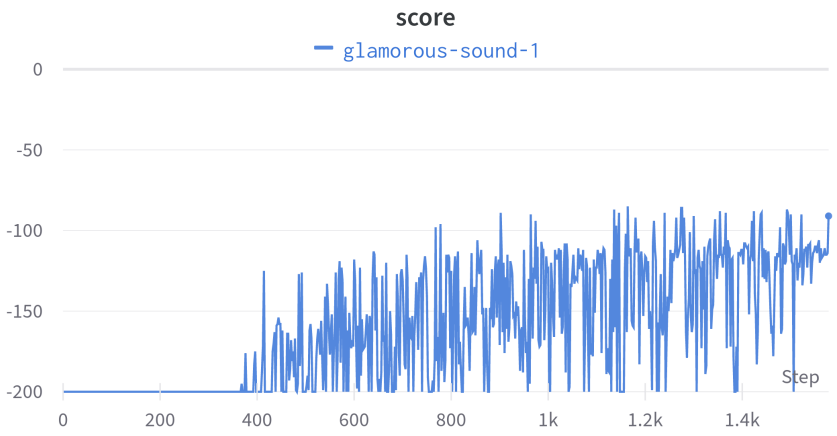


Figure 5:

BUFFER_SIZE	10000
BATCH_SIZE	256
GAMMA	0.99
LR	0.002
UPDATE_EVERY	48
POLICY	eps
LAYERS	7

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.

Sathvik Joel K

Hyper parameter Tuning

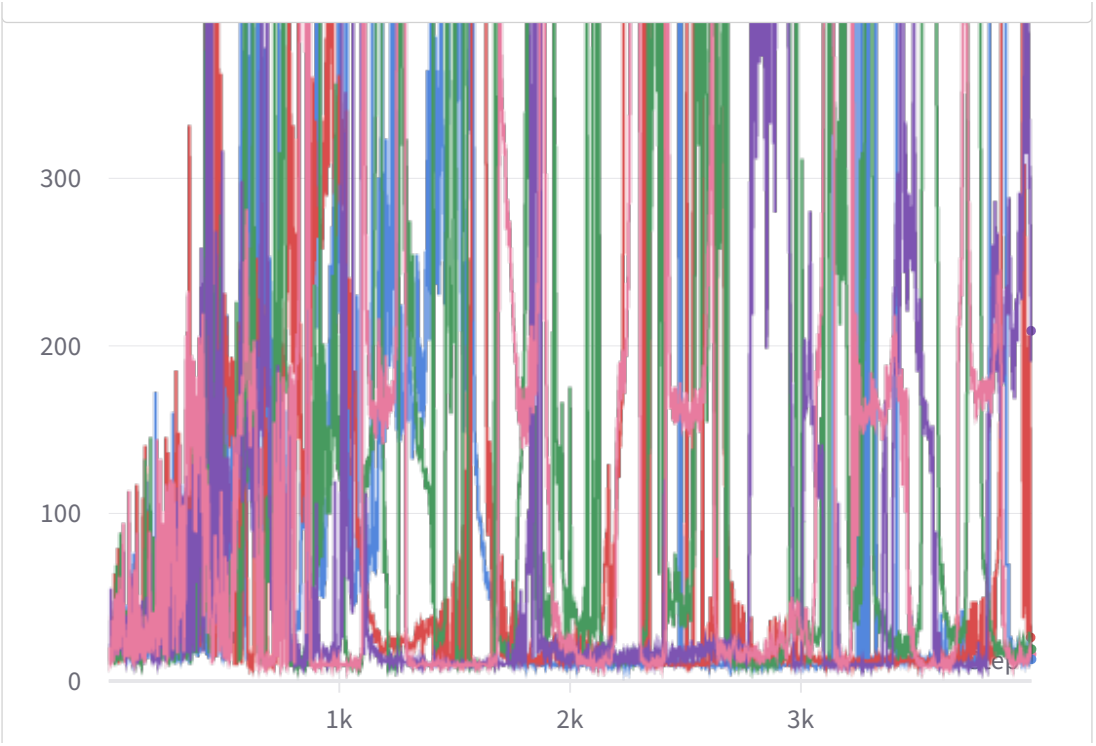
Criterion

We say that the agent has been trianed if the average score across 100 runs is greater than 475

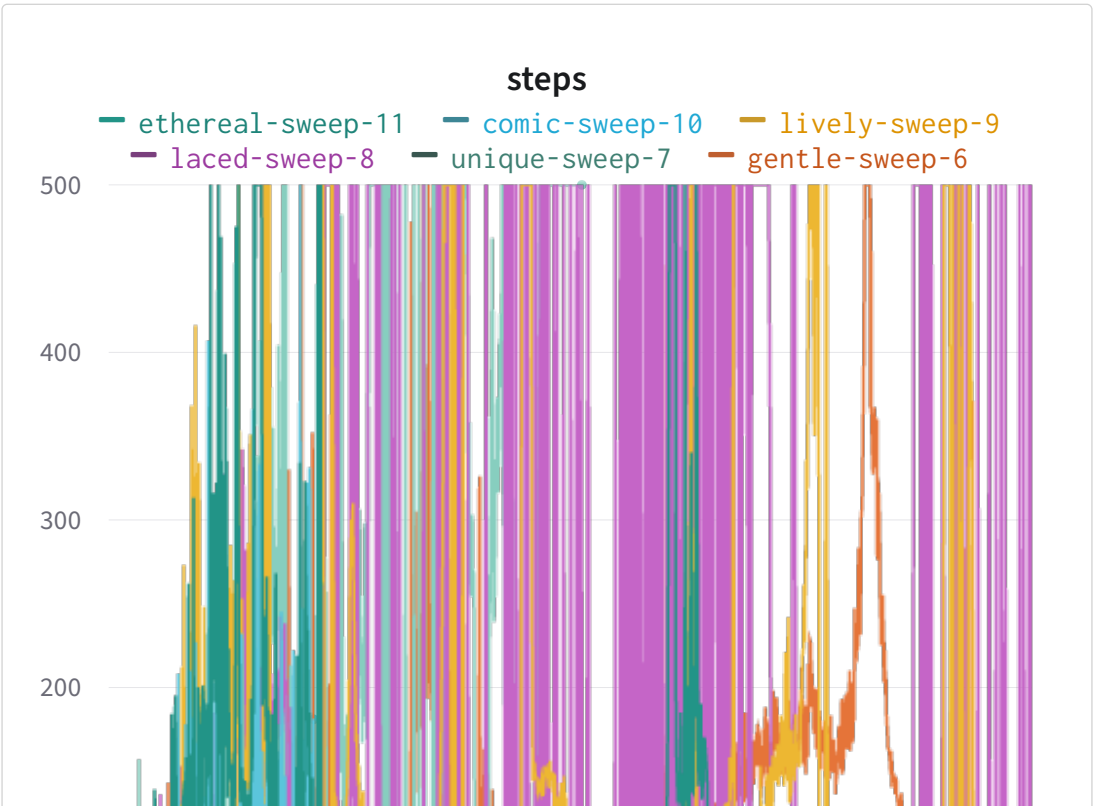
Random Sweep

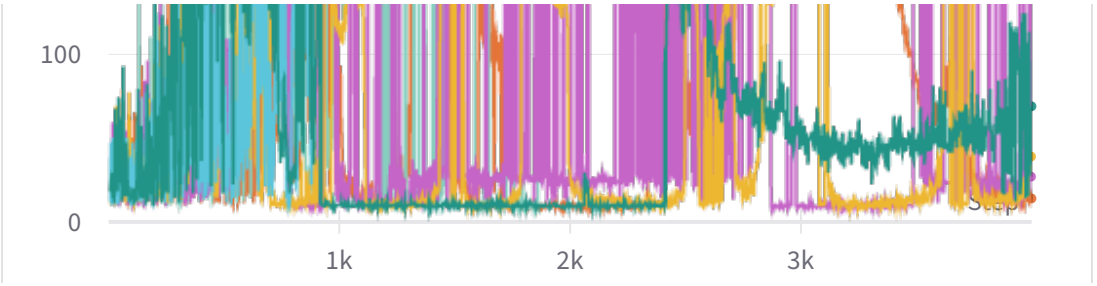
Our Approch is to first run random sweep until we find a good set of Hyperparameters after which we try to optimize around it.

All of the runs were run until 4k Episodes. Here are reward curves for few of the sweeps we made (All the sweeps are not included)



Import panel Add panel





Import panel Add panel

Positive Sweep

After more than 20 runs, we found a Hyper Paramters combination that converged.

batch size = 128

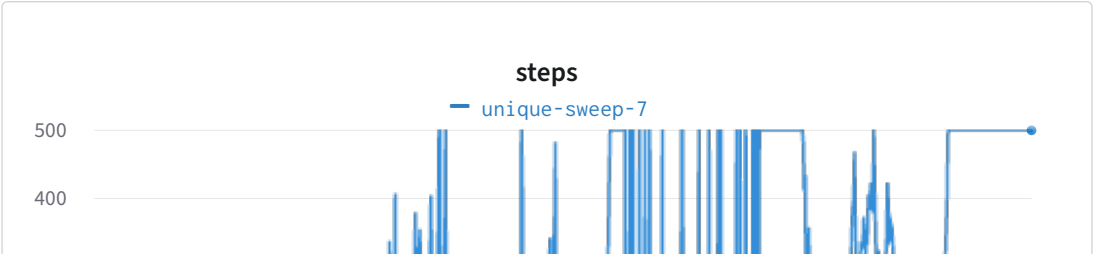
buffer size = 100000

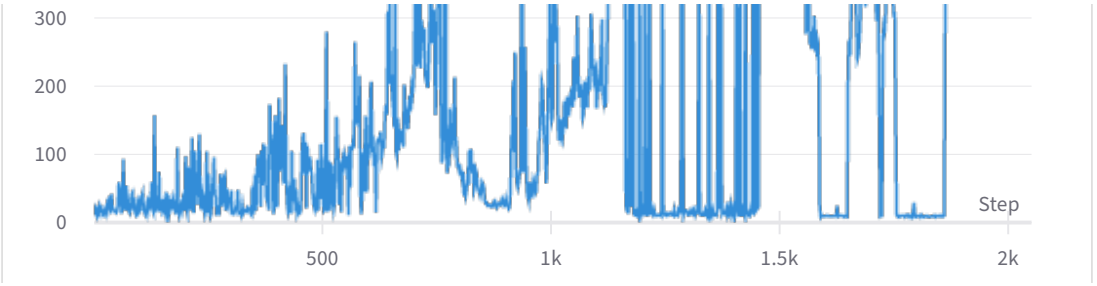
gamma = 0.9985

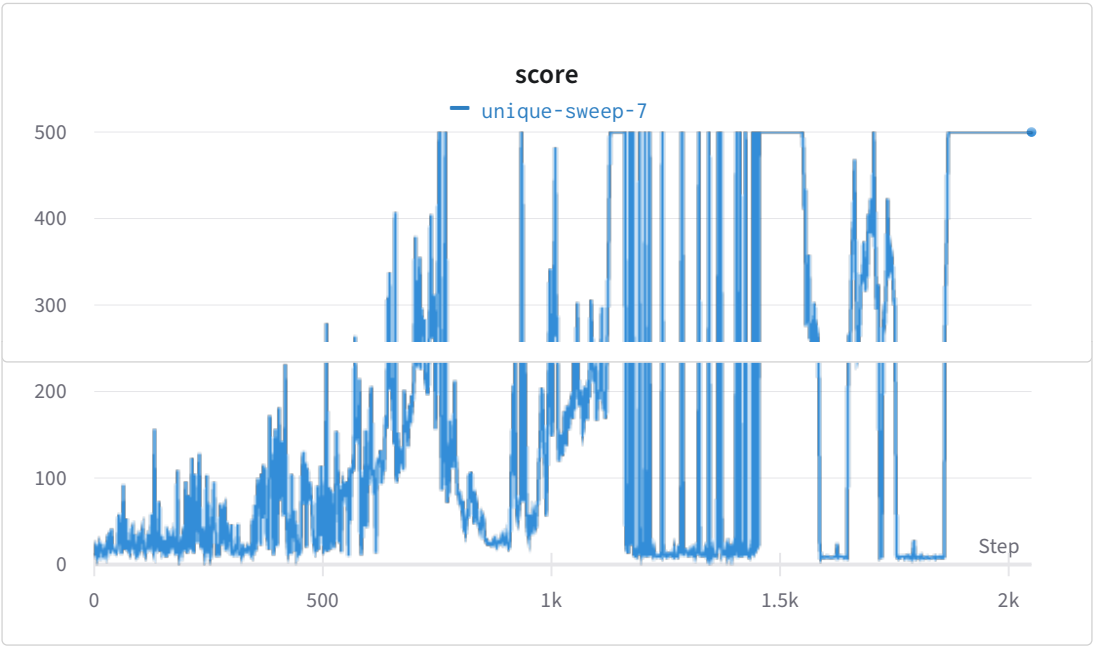
learning rate = 0.032

update every = 40

Layers = 3







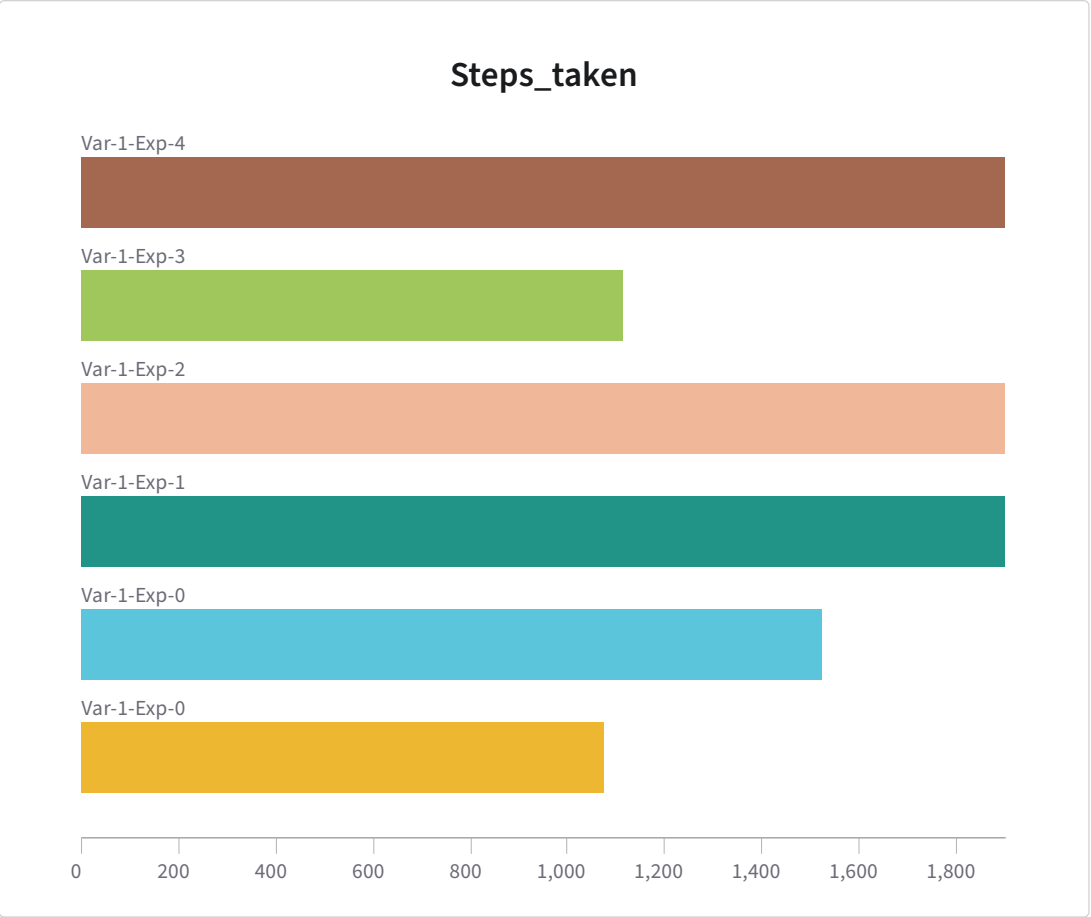
Import panel Add panel

Increase the Number of Layers to 7

We tried to increase the number of layers to 7

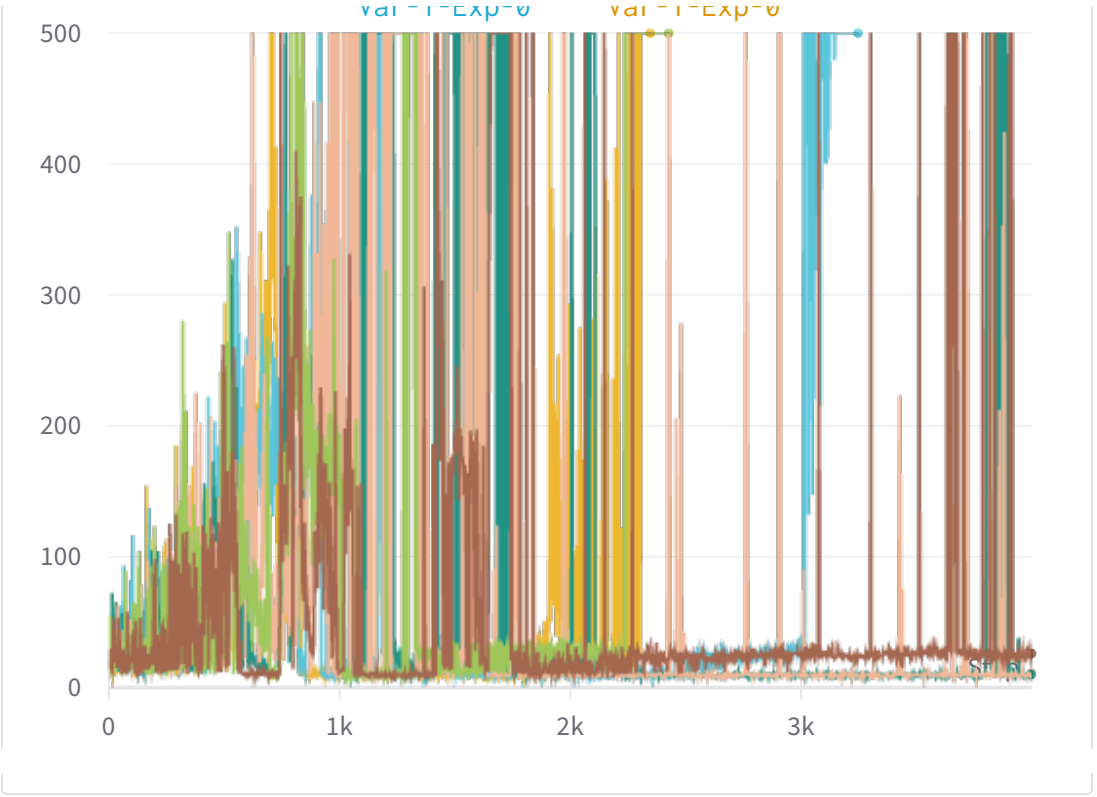
Here are the results

fc1_units=128, fc2_units=256, fc3_units = 512, fc4_units = 256, fc5_units = 128, fc6_units = 64



Import panel Add panel

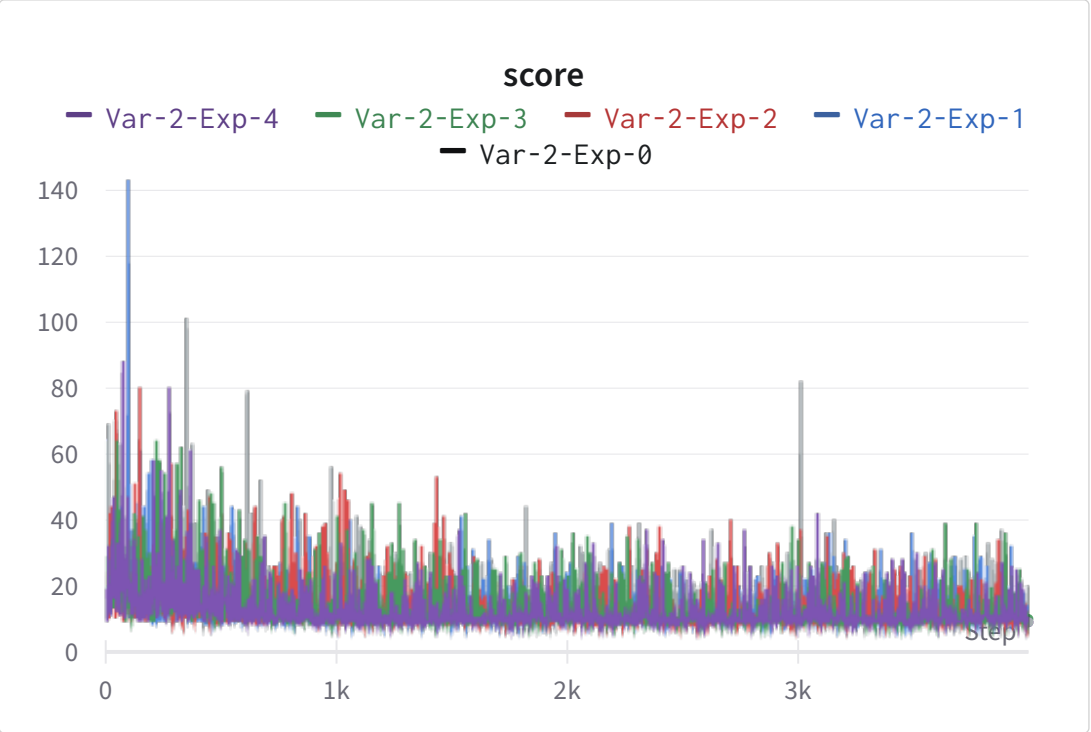
Here are the reward Curves



Import panel Add panel

Decrease the Number of Layers to 5

fc1_units=128, fc2_units=256, fc3_units = 512, fc4_units = 256, fc5_units = 128

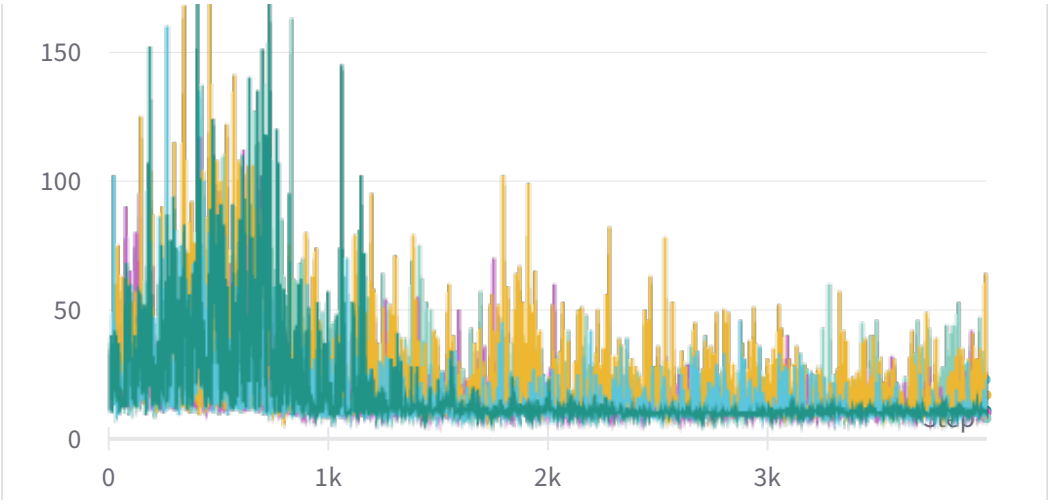


Import panel Add panel

The performance dropped. Clearly.

Increase the batch size

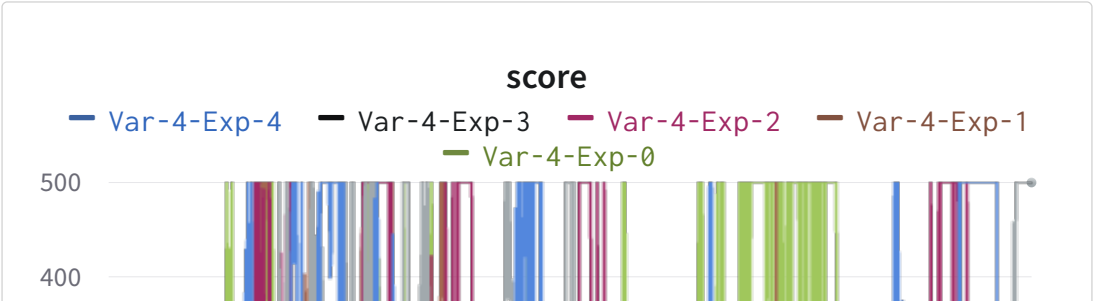
We increased the Batch size to 512, but the results didn't seem to increase.

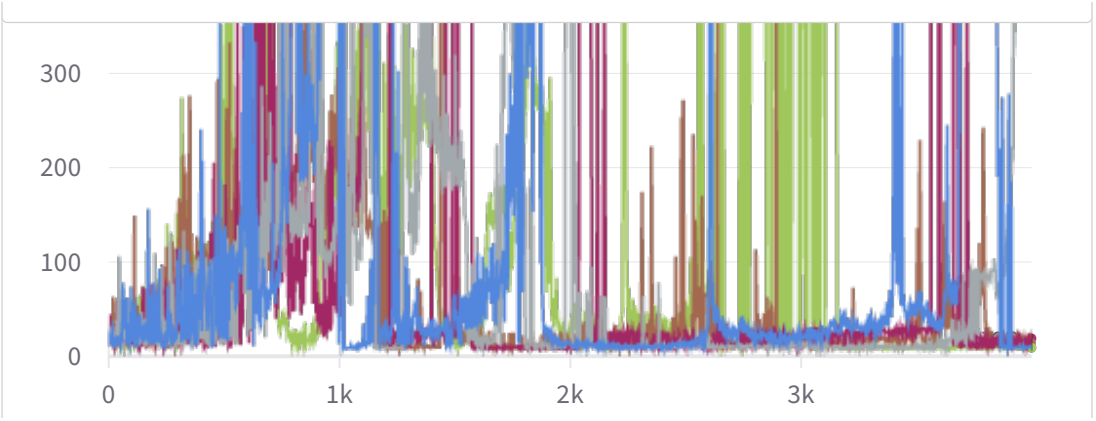


Import panel Add panel

Increase the Learning Rate

We increased the leraning rate but the performance incresed from the above two cases but still it didnt cross the case with 7 layers.





Import panel Add panel

Conclusion

So, the case with 7 layers is performing the best.

Here are the complete hyperparameters

batch size = 128

buffer size = 100000

gamma = 0.9985

learning rate = 0.032

update every = 40

Layers = 7