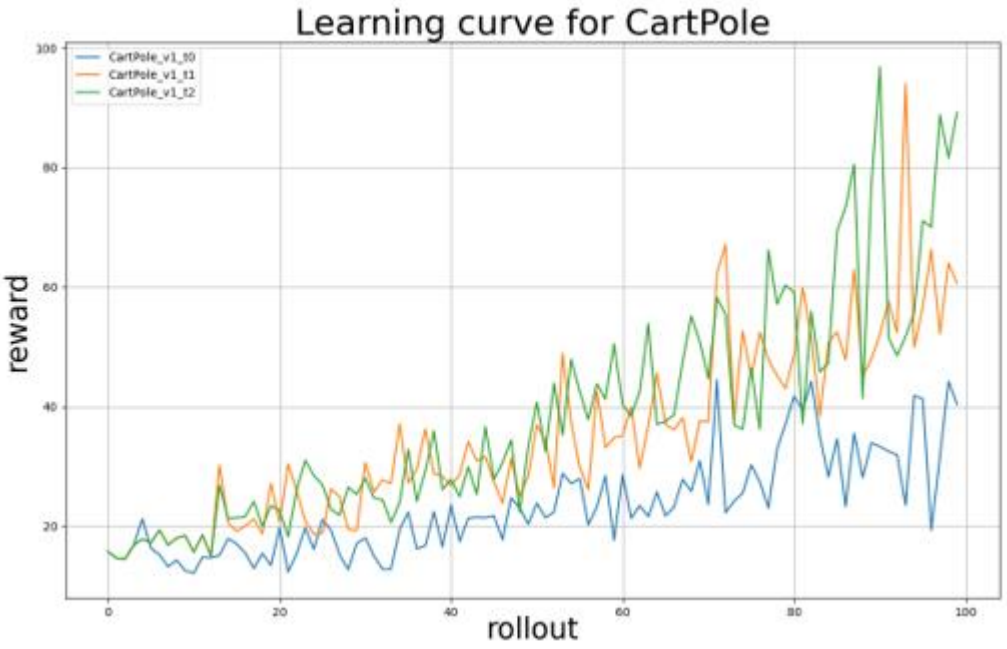
Report – Santhosh Vanamala -1001964729.

Link - https://github.com/SanthoshVanamala/Policy-Gradient-Assignment.git

1.
```python
def estimate_loss_function(self, trajectory):
    loss = list()
    for t_idx in range(self.params['n_trajectory_per_rollout']):
        # TODO: Compute loss function
        # HINT 1: You should implement eq 6, 7 and 8 here. Which will be used based on the flags set from the main function
        _log_probs = trajectory['log_prob'][t_idx]#extracts log probabilities by agent at each time stamp
        r = torch.tensor(trajectory['reward'][t_idx],dtype = torch.float32, device = get_device())#this extracts rewards by using pytorch sensor

        r -= r.mean()#subtracts the mean of rewards from tensor rewards
        r/=(r.std() + 1e-8)#this scales rewards by standard deviation
        actn_loss = (-_log_probs*r).sum()#calculates action loss
        loss.append(actn_loss)#append action loss to loss
    loss = torch.stack(loss).mean()
    return loss#return loss
```

2.




3.discounted- reward based Policy Gradient T2 performs best in all 3 versions of Cartpole-v1 environment.

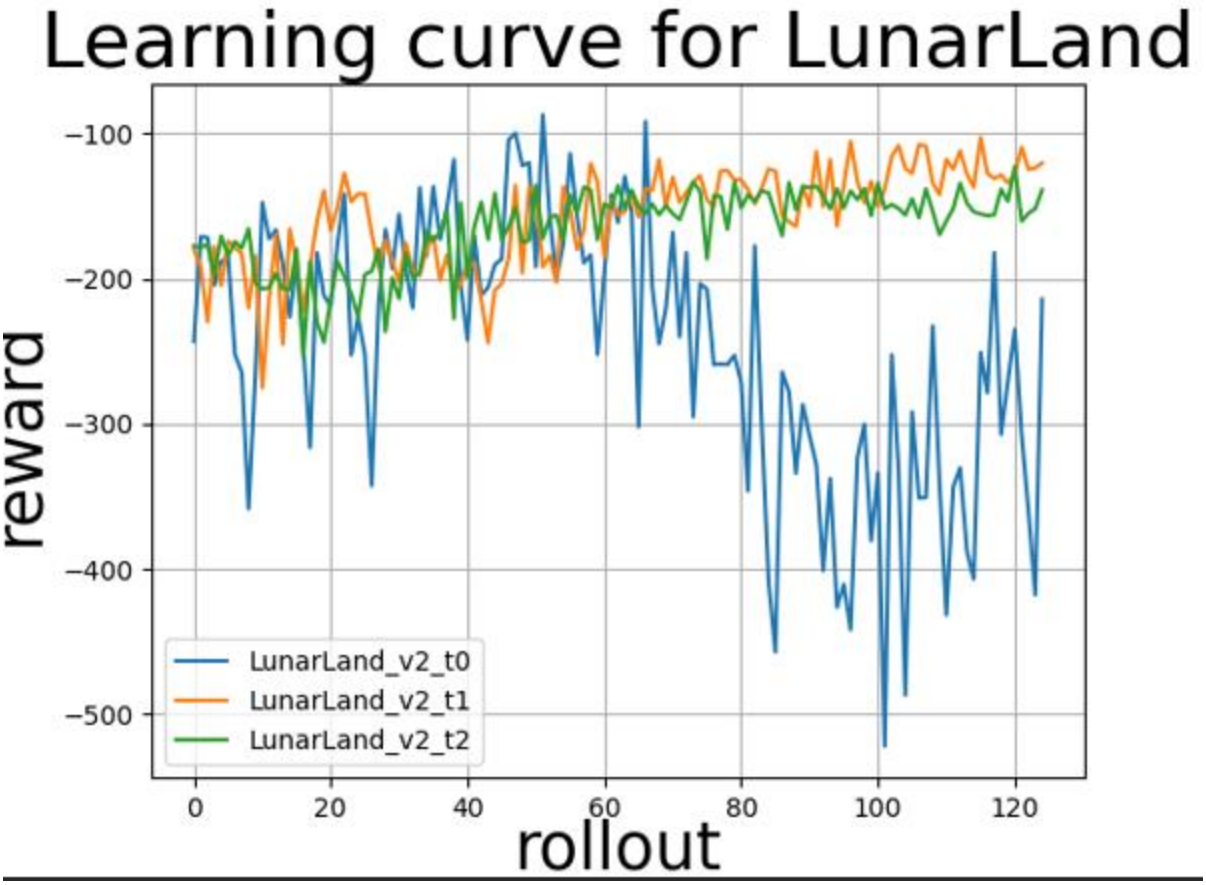
The discounted-reward-based variant takes into account both current and future benefits, but gives the current rewards greater weight, which may account for its higher performance. The agent becomes more focused on adopting behaviors that provide current gains when future rewards are discounted, which might improve policy learning in this particular context. Discounted incentives can also aid in stabilizing learning by avoiding the accumulation of big rewards, which could amplify gradients.

The reward-to-go based version, on the other hand, considers the rewards from the current timestep until the conclusion of the episode whereas the return based version merely concentrates on maximizing

the aggregate of rewards. In order to develop a policy that emphasizes immediate rewards and stabilizes the learning process, these two variants might not be as successful as the discounted-reward-based one.

Learning performance may be impacted in a variety of different ways when the number of trajectories per rollout rises. In general, more trajectories per rollout might result in greater learning performance since they provide the agent more material to learn from, enabling it to investigate the environment more effectively and perhaps come up with better solutions. Nevertheless, the precise effect might change based on the job and learning algorithm being employed.

1. Exploration vs. Exploitation: The agent may explore more of the environment and learn more about the effects of its choices when there are more trajectories per rollout. As the agent learns better action sequences via exploration, improved policies may result. The agent may spend a lot of time investigating less-than-optimal behaviors before settling on a suitable policy, which might delay learning. For effective learning, exploration and exploitation must coexist in harmony.
2. Variance reduction: The variance of the calculated gradients utilized in the Policy Gradient method can be decreased by increasing the number of trajectories each rollout. Lower variance can result in more stable learning and better convergence since the gradients won't be as influenced by random environmental variations. In situations with stochastic dynamics, where individual trajectories might have a lot of variation, this can be especially useful.
3. Computational complexity: Each iteration of the learning method requires more computing power as the number of trajectories per rollout rises. Longer training periods result from adding more trajectories, which might be problematic if computer resources are limited.




2. Increase in trajectory may increase in accuracy of value function. Which can lead to better performance