Exercise 5

Amr Alkhashab - 905833
ELEC-E8125 - Reinforcement Learning

October 23, 2020

1 Task 1

1.1 (a)

For this part, the discounted reward, sigma and $weighted_probs$ that are used are:

```
#sigma is the sqrt of the variance, and normal function takes
    std not variance.
  sigma = math.sqrt(variance)
  discounted = discount_rewards(rewards, self.gamma)
  weighted_probs = -action_probs * discounted
  loss = torch.mean(weighted_probs)
```

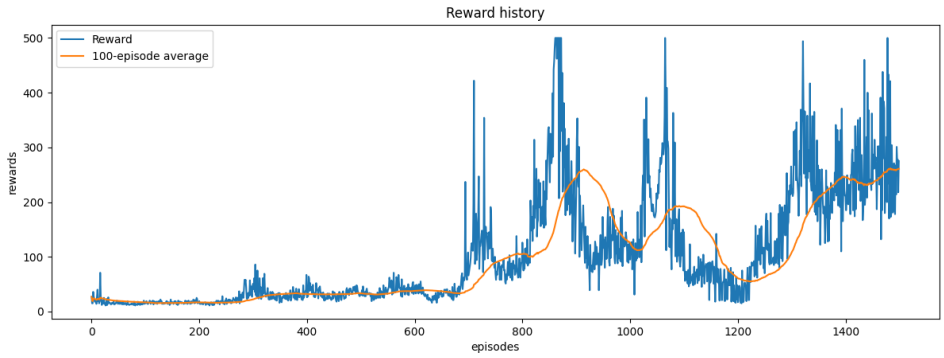

Figure 1: Training performance for single episode for Task1a

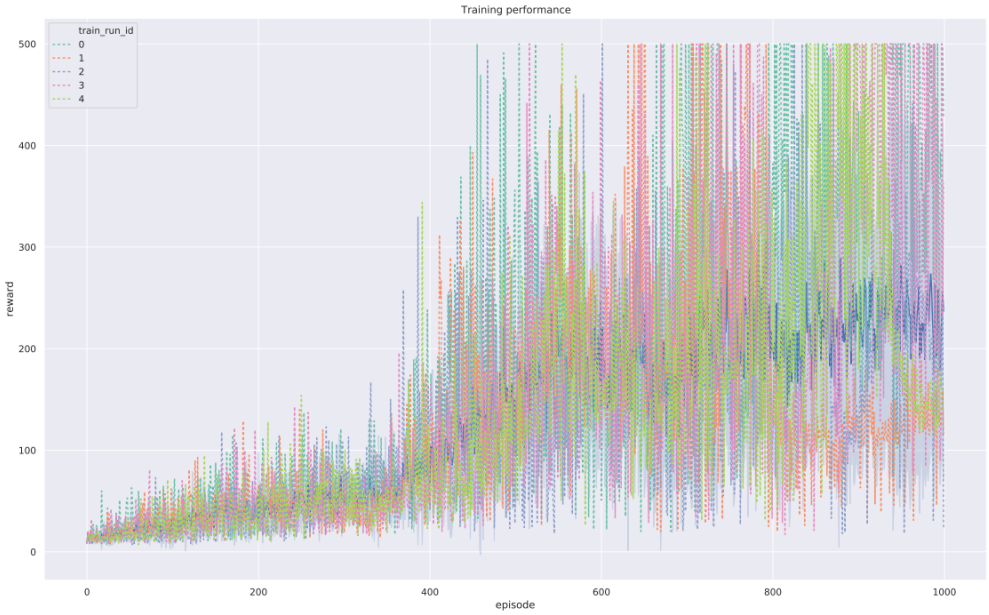

Figure 2: Training performance for single episoder for Task1a

1.2 (b)

The change part was:

```
1    weighted_probs = -action_probs * (discounted - 20)
```

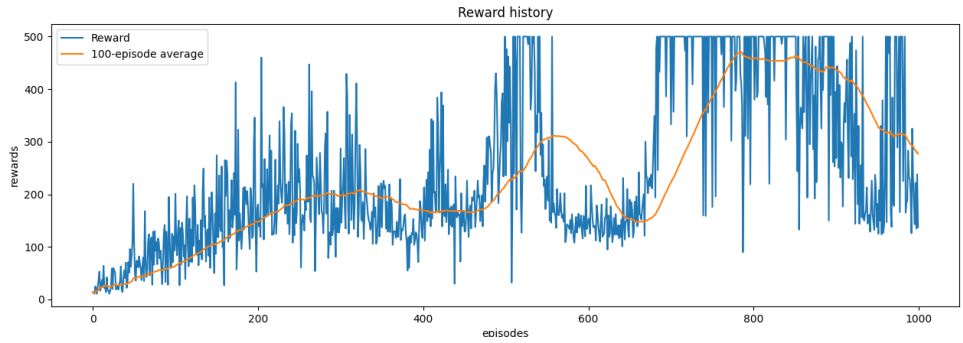

Figure 3: Training performance for single episoder for Task1b

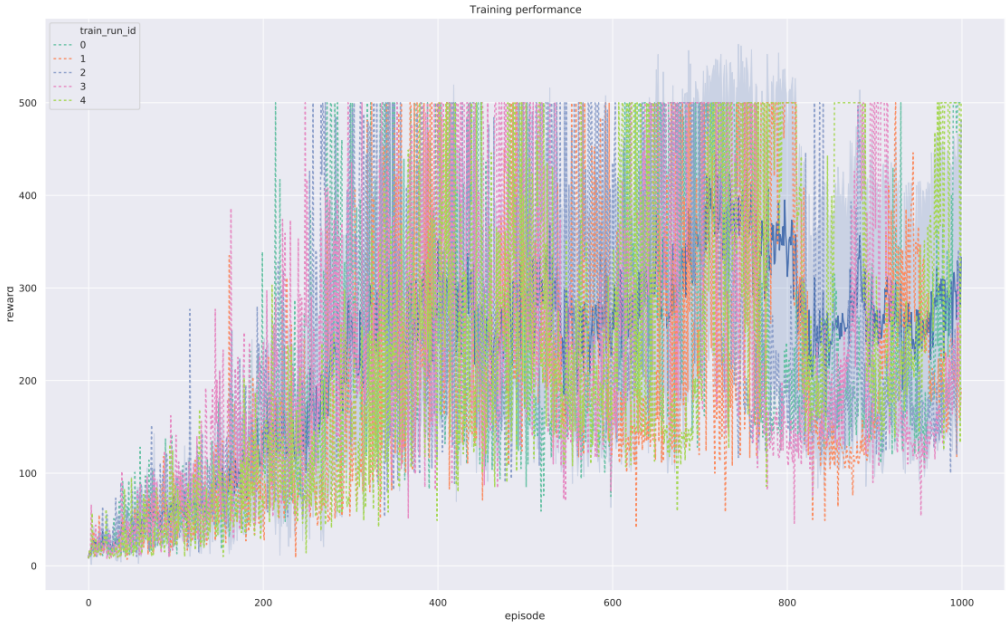

Figure 4: Training performance for single episoder for Task1b

1.3 (c)

The discounted reward is modified to:

```
discounted = discount_rewards(rewards, self.gamma)
      discounted -= torch.mean(discounted)
      discounted /= torch.std(discounted)
```

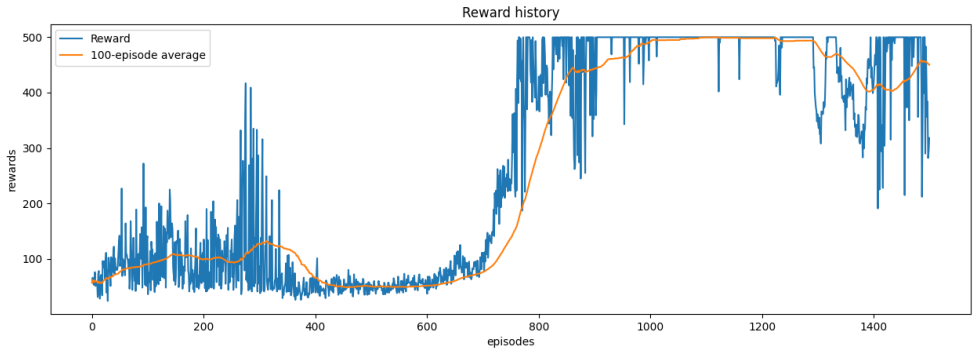

Figure 5: Training performance for single episoder for Task1c

2 Question 1.1

Conditions of good baseline [1]

1. the baseline can be any variable or function, as long it doesn't change with each action, in a sense that it will not change our gradient and introduce any unwanted bias.

2. obtained through approximation over many trials, that lead to obtaining Optimal baseline that minimizes the variance.

3. The baseline should change for each state, since some state have high values, while others have low value. For, high values action, high baseline is needed to differentiate them from lesser ones, and the same can be said for low values.

4. a good choice of baseline is $\hat{v}(S_t, w)$, which means if the return is higher than this baseline (current extimates), the probability increases of performing this action, and if below the probability decreases. This is done, by moving in either the positive or negactive direction of the gradient. The desired value is then the the mean expected return in that sense, one example can be represented as b_h.

The optimal baseline for Reinforce which has no value function, where the approximation is done over many trials of the return:

$$b_h = \frac{E_\tau\left[\left(\sum_{t=0}^{H} \nabla_{\theta_h} \log \pi_\theta(a_t|s_t)\right)^2 R_\tau\right]}{E_\tau\left[\left(\sum_{t=0}^{H} \nabla_{\theta_h} \log \pi_\theta(a_t|s_t)\right)\right]^2}$$

3 Question 1.2

The baseline can reduce the variance significantly and thus improving the training speed and the rate of convergence.

4 Task 2

```
# first case
variance = math.pow(self.sigma0, 2) * math.exp(-self.c *
    episode_number)
sigma = math.sqrt(variance)
#second case sigma was used as variance without sqrt to obtain
    same result as indicated below.
self.var = torch.nn.Parameter(torch.tensor([10.0]))
sigma = self.var
#the std one
#sigma = torch.sqrt(self.var)
```

The plot here are done in 10 episodes only:
The results:

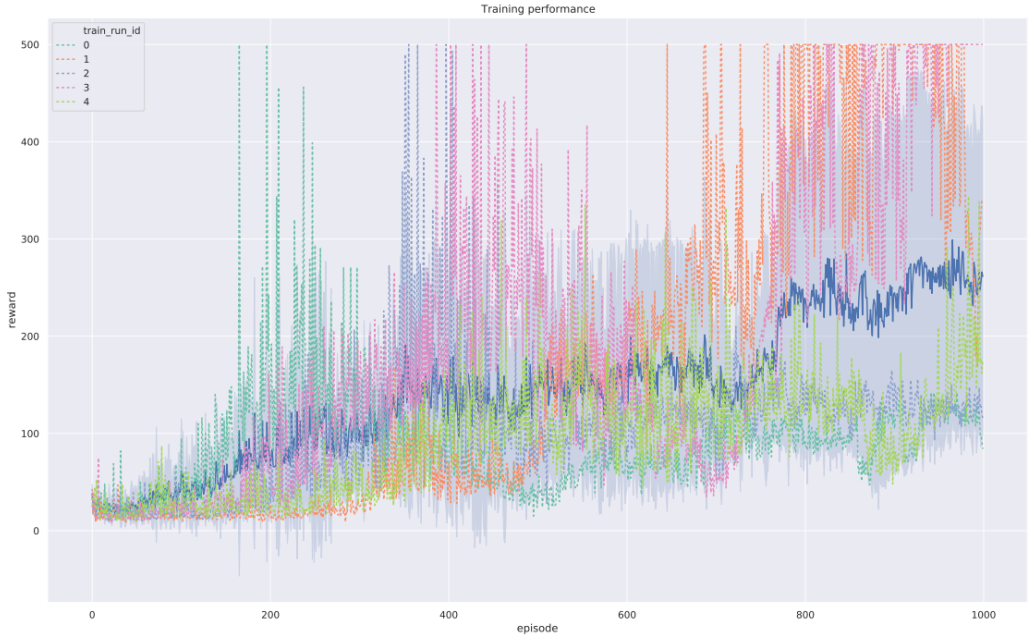

Figure 6: Training performance for10 episoder for Task2 exponentially decay

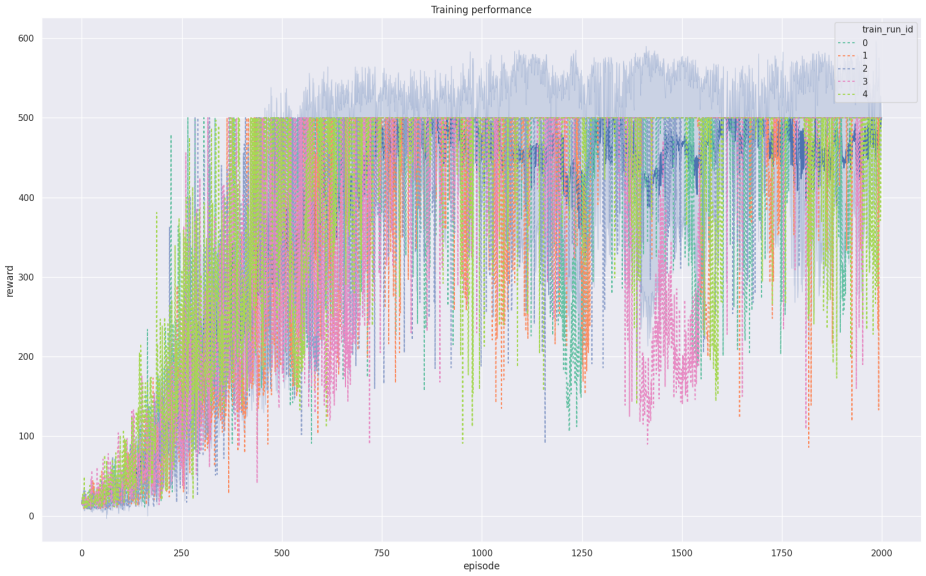

Figure 7: Training performance for 10 episoder for Task2 variance learned

5 Question 3.1

At the begining of the training the Task1 has a constant variance of 5, while exponential one and learned variance had variance of 10. Larger variance means more exploration at start. For constant variance will always explorate, but exploration will never decrease. It simple method that uses low computational resources. For decaying variance, the agent is encourage to explore at the start, then as the number of episode increases it explore less and less. This process is not done autonmously, and need some testing to adjust appropriate constants, however, here we can ensure our desired level of exploration and exploitation of the agent will be satisfied. For a learning variance, the agent try to find the variance on it's own, either by increasing it for more exploration or decreasing it by the time when the certainity over the right action increases, the system tend to adjust that autonomously, without many input from the user. The system may choose to increase explore or decrease it based on the optimization. It can stuck in bad local regions, so desired level of exploration is not ensure most of the time, as other method, this is affected by the intial value.

6 Question 3.2

Choosing starting large number in this case is always the best, since it encourage exploration in the start and that allow for faster training and then variance decrease by the time when certainity increases. However, choosing small value will lead to bad results. since there will be less exploration, and less range of action to choose from, and the loss function will optimize based on such data and this result in slower training . Based on observation, the vairance change very slowly from the intial state, thus the varaince will stay low for most of the time and this will likely increase the learning time alot. Choose a value near zero is also bad, since variance can change to a negative value, and that result in error, since variance cannot be a negative. (softplus can be used in such case to prevent such error).

7 Question 4.1 and 4.2

No, it cannot be implemented directly, because our implementation is based on on-policy policy gradient, not off-policy. No batches of observation, action and reward are stored in memory. To modify our previous code, first, since we only have one policy, we have to introduce now two policy, target policy (the learning policy) and the behavior policy that select actions. Then use the sample avearage to adjust the new expectation of our distrubution when data is sampled from another, this is needed to modify the old loss function. Our function will be weighted by the samplying ratio term. Then we start storing value in memory and do batches as in previous exercise.

$$E_{\tau\sim\pi'(\tau)}\left[\left(\prod_t \frac{\pi_\theta(\tau)}{\pi'(\tau)}\right)\left(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_t r_t\right)\right]$$

8 Question 5.1

In our implementation, the only way to control the behavior (actions) is by adjusting reward function, by a bound on the max and min mean action using if condition for example or terminating episode once certain action is achieved. If unbounded action and unbounded reward, our model during training can produce value of action greater than the absolute of system action range [-10, 10], and in some case where there might be a bug the absolute action values can increase rapidly towards infinity. If we assume that this value correspondes to the amount of voltage that going to be supplied to the motors, then the motor will fail and there will be permenant damage, or the cart could learn to balance, but at high velocity, it could containoulsy hit surrounding object and fails. In our physical world there are many parameters that are not considered in our simulation like inertia for example. We assume, if the teminate state is reached the system reset automatically, that not the case in a physical world, where if the system turns off, the system could still move and hit surround object by mistake due to inertia, and as the speed and forces increase the more severe the situation which is resulted from unbounded reward and action.

9 Question 5.2

If a hard limit on the actions, which could be an if condition or introduction of a new termination states to hard limit the action couldnot be used, then, We could penalize the system with negative reward proportional to how much the the action is when out of bound. The higher the action, the higher the negative reward, with such reward the agent will learn to not increase the action so much. A second solution, is design rewards, which allow achieving the objective within coinstrained observation (bounded reward), in a way that a high action value will always yeild bad results. In the case of catapolt for example, we can try to teach the agent to balance the rod, under certain speeds at a certain fixed position by carefully designing our rewards functions, so that it learn to do the objective at low action values. This can also be affected by the initial value, it should be set in a range around the true value, for example If the intial action value are zeros, then the agent will try to find the optimal action arround that point to maximize rewards and thus won't have to go further away. We can also decrease range of exploration, using lower sigma so that we don't sample high action values. Using bootstrapped approach is prefered in such case, to adjust the policy in each timesteps without having to wait for the epiosde to end.

10 Question 6

Yes, it can be used with discrete action space as in exercise one, because policy gradient only depend on commuting the loss for functional approximation, this functional approximation can be eaily modified to discrete action , meaning that all part of the algorithm is compatiable with the change to the discrete vesion, changing the outter layer from one node (containous action) to serveral nodes equat number of discrete actions, then applying softmax will solve all existing problems. In our implimentation, the out layer had only one node, in discrete space, it will have nodes equals to the number of discrete actions. Those action will be

passed to softmax layer, and then when evaluating take the action with highest probablity, else sample from distribution using distribution,Categorical.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.