Typically, adversarial learning such as GAN is formulated such that the discriminator tries to maximize the "real data" probability and minimize the "fake data" probability. In this formulation, we assume that the generator tries to maximize the "fake data" probability.

But this is not the only way of formulating adversarial learning. You can do it in the other way around as well. i.e. the discriminator tries to minimize the "real data" probability and maximize the "fake data" probability, whereas the generator tries to minimize the "fake data" probability.

The most important condition here is that the discriminator and generator should be in competition with each other.

Taking those ideas to the task 2 in mandatory assignment 3, we can see that the following two adversarial approaches are equivalent:

1. Discriminator tries to maximize P(expert-traj) and minimize P(policy-traj) whereas the policy tries to maximize P(policy-traj)

2. Discriminator tries to minimize P(expert-traj) and maximize P(policy-traj) whereas the policy tries to minimize P(policy-traj).

where P(expert-traj) = probability of the expert trajectory and P(policy-traj)=probability of the policy generated trajectory.

Further, note that (line 278) the reward of the PPO is defined by reward = -P(policy-traj) (Note the minus sign. See also the function get_reward() and its call in line 364). That means that when PPO is trained, it tries to maximize the reward and hence it (policy) tries to minimize P(policy-traj) . Therefore the relevant case is number (2) above. In case number (2), P(expert-traj) should be minimized (i.e. targets should be zeros) and P(policy-traj) should be minimized (i.e. targets should be ones).

Therefore the line 297, should be completed as follows:

```
self.model_prob.train_on_batch(all_ob_ac,[self.zeros, self.ones])
```

where `self.zeros` is the target for expert trajectories and `self.ones` is the target for the policy generated trajectories.