ECE 276 Assignment 3 : Policy Gradients

November 1, 2020

The goal of this assignment is to implement different policy gradient methods for environments with discrete and continuous action spaces. In the first part of this assignment you will solve the CartPole-v1 environment in gym and for the second part you solve a custom reach task for the 2 link arm.

Question 1 - CartPole-v1

For each of the following questions, implement the policy using a fully connected neural network. Since the action space of the environment is discrete, the policy should return a categorical distribution over the set of actions. For question 1,2 and 3 train the policy using a batch size of 500 steps, for 200 iterations of policy updates. For training set $\gamma = 0.99$ and for evaluating the average returns use the data collected for training.

1. Implement a vanilla reinforce algorithm given by the following gradient update for your policy.

   $$\nabla J(\theta) \approx \frac{1}{n} \sum_{i=0}^{n} G(\tau) (\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)) \tag{1}$$

   Where $G(\tau) = \sum_{t=0}^{T} \gamma^t r(t)$ is the Monte Carlo estimate of the discounted return, $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t given s_t and n is the number of episodes. Plot the average returns at each iteration.

2. As discussed in class, the gradient at step t is only dependent on the future rewards, hence we can modify the policy gradient as follows:

   $$\nabla J(\theta) \approx \frac{1}{n} \sum_{i=0}^{n} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} \gamma^{t'-t} r(t') \tag{2}$$

   Implement the policy gradient algorithm using the above update rule. Plot the average returns at each iteration. Compare your results with question 1.1.

3. To reduce the variance of the estimated returns, subtract the returns using a constant b such that the mean of the modified returns is 0. It is also observed empirically that dividing the modified returns with the standard deviation of the estimated returns improves training. Implement the policy gradient with these modifications.

   $$\nabla J(\theta) \approx \frac{1}{n} \sum_{i=0}^{n} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) (\sum_{t'=t}^{T} (\gamma^{t'-t} r(t')) - b) \tag{3}$$

Plot the average returns at each iteration. Compare your results with question 1.1 and 1.2.

4. Train the policy with the update in question 1.3 with following batch sizes - $\{600, 800, 1000\}$ for 200 iterations. For each batch size, plot the average returns in each iteration. Does increasing the batch size improve training?

Question 2 - 2 Link Arm

NOTE: Before starting this assignment, download the folder modified-gym-env, and install the package using pip install −e modified−gym−env/ using the same virtual environment as assignment 1. This will give access to the modified reach environment using

```
env = gym.make(”modified_gym_env:ReacherPyBulletEnv−v1”)
```

The goal of this question is to train a policy using policy gradient method, to move the arm from the initial position to target position using policy gradient method in less than 500 iterations. Since the action space is continuous, implement a policy that samples from a multivariate normal distribution. ie $a \sim N(f(s), \Sigma)$, where $f(s)$ is a neural network and Σ a diagonal covariance matrix. Try to find the smallest possible batch size that succeeds.For training, set $\gamma = 0.9$. Plot the average return for each iteration for your final model. Also upload a gif of your final policy completing the task for a single episode on canvas(you can use screen capture software to record the videos).
Expected result: The final policy should be able to reach the target location for the randomly initialized reach task.

Note

Here are a few suggestions that you could use to complete the assignment:

1. stochastic policies - To implement the stochastic policies, you can make use of the torch.distributions library. The library has inbuilt functions to sample from the distribution and return corresponding log probabilities of sampled points.

2. Implementing stochastic policies for continuous action space - To implement the Gaussian policy you can use the Multivariate Normal function class from the torch.distributions library. The mean of the Gaussian should be a function of your state and the covariance matrix can be a diagonal matrix initialized with random value. Also make sure that while training, your eigenvalues of your covariance matrix does not become less than 10^{-3} and always remains positive.

3. For question 2, you can fix the robot arm initial position by passing the argument rand_init=False in the gym.make command. This is a simpler environment to train. **You do not need to plot the performance of this environment in your report**. This is provided for you to check your implementation.

4. Seed values - For question 2, train with different pytorch seed values to make sure your code is working.