Name: Manoj Aryal
ID: 2020470025

CartPole-v0

Problem Description:

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.
(source: https://gym.openai.com/envs/CartPole-v0/)

1) **Observation Space:**

There's 4 observation space in this environment. They are Cart Position, Cart Velocity, Pole Angle, Pole Velocity at Tip. Our agent will take certain action A(t) at state S(t), time step t and get the reward of R(t). And by interacting with these spaces by taking the actions that gives the best reward, the goal of our agent is to solve the environment.

Type: Box (4)

Observation Space	Min	Max
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	~ -41.8°	~ 41.8°
Pole Velocity At Tip	-Inf	Inf

2) **Action Space:**

There's 2 discrete action space in this environment. They are 'Push Cart to the left' and 'Push Cart to the right'. Our agent will pick the action A(t) at time step t from the current state S(t) so as to receive the reward R(t) and next state S(t+1) from the environment.

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

3) Policy Gradient:

Policy gradient methods are model free reinforcement learning technique that directly model and optimize the policy rather than first predicting the value function.

With this method, we can obtain the best policy directly for the environment. There are several Policy Gradient Algorithms like Actor-Critic, REINFORCE, Proximal Policy Optimization, Deterministic Policy Gradient, etc.

The policy is modeled with a parameterized function respect to θ, $\pi\theta$ (a|s) and the reward function will depend on this policy. Then we can optimize θ to get the best possible reward using various algorithms.

One of the most popular Policy Gradient algorithms REINFORCE algorithm is described as:

REINFORCE:
For this algorithm:

i) We first initialize θ arbitrarily.
ii) Collect the trajectories using current policy
iii) Store log probabilities of the policy and reward value
iv) Update the policy parameter by performing policy gradient with calculated discounted future reward
v) Repeat step ii), iii) and iv)

4) Which method (on-policy or off-policy) is used in this experiment? And what is the difference of on-policy and off-policy?

A code snippet from homework.py

from tianshou.trainer import onpolicy_trainer

So, for this particular experiment, on-policy was used.

On-Policy: In this policy, the action-value function $Q(s,a)$ is learned only from the action that we take using the current policy $\pi(a|s)$.

Example: SARSA algorithm where it updates Q-value using the Q-value of the next state and the action using current policy.

i.e $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma Q(s',a') - Q(s,a))$, where a' is taken using current policy.

Off-Policy: In this policy, the action value function $Q(s,a)$ is learned from taking different random actions (does not follow the current policy to take the action at next step).

Example: Q-learning where it updates Q-value using the Q-value of the next state and the greedy action.

i.e $Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max a' Q(s',a') - Q(s,a))$, where a' is action that gives the maximum value

Experiment Result:

After Ep:1, 178/1000
Final reward: 200.0, length: 200.0 was obtained and the environment was solved.

On-policy algorithm was used with the **discounted factor (gamma) of 0.9.**
The policy was saved in log folder as 'policy.pth'

With **discounted factor of 0.1**, the model was not able to solve the environment even after lots of episodes.

And with **0.5 discounted factor**, the model solved the environment in Ep:1 284/1000 timesteps.

So, we can tell as you reduce the value of gamma, the model finds it difficult to solve the environment. Saying this, it is not optimum to put the value of gamma as 1 either because that would mean our agent will care about all the future reward equally and try to maximize the sum.

Among several value of gamma, 0.9 worked really well.

I tried putting the value of discount factor (gamma) as 0.5 and saved the python file inside submission/code folder.

Bonus:

The Effect of Discount factor:

Discount factor plays a huge role in model efficiency as it is the key point for 'exploration-exploitation' tradeoff in reinforcement learning domains.

The discount factor determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.

So, with $\gamma = 0$, our agent will only care about the immediate reward and produce actions accordingly and if $\gamma = 1$, then our agent will produce each action so as to produce maximum total reward in future.

Discounted reward is given as:

$$G(t)=R(t)+\gamma R(t+1)+\gamma\text{^}2\ R(t+2)+\ldots.$$

If γ is large, then agent will consider irrelevant information to produce an action which will hinder our model and if γ is small, then agent will only care out the immediate value and our model will be very unstable for the long run.

So, we should be very thoughtful about picking the value of γ as it vastly influences our result.

Conclusion:

So, Reinforcement Learning is the area of machine learning where an agent tries to maximize its reward by taking optimal action interacting with the environment. After the DeepMind's success, deep learning revolution with very high computational power devices, it is gaining huge popularity.

And I really believe this field will help a lot to automate things especially in self driving cars and robots.