FPT Education

FPT UNIVERSITY

Actor-Critic for Continuing Tasks

FPT Education
FPT UNIVERSITY

Objectives

- Derive a sample-based estimate for the gradient
- Understand the Actor-Critic algorithm

Estimating the Policy Gradient

- Definition:
  - Estimating the policy gradient is a crucial step in policy gradient algorithms
  - It allows us to update the policy parameters in the direction that maximizes the expected return.

FPT Education
FPT UNIVERSITY

Estimating the Policy Gradient

- The gradient of the average reward:

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$$

- We simply make updates from states we observe while following policy pi.

$$S_0, A_0, R_1, S_1, A_1, \ldots, S_t, A_t, R_{t+1}, \ldots$$

Estimating the Policy Gradient

- This gradient from state St provides an approximation to the gradient of the average reward.
- The stochastic gradient descent update looks like for the

$$\nabla r(\pi) = \sum_s \mu(s) \sum_a \nabla \pi(a \mid s, \boldsymbol{\theta}) q_\pi(s, a)$$

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \sum_a \nabla \pi(a \mid S_t, \boldsymbol{\theta}_t) q_\pi(S_t, a)$$

$$S_0, A_0, R_1, S_1, A_1, \ldots, S_t, A_t, R_{t+1}, \ldots$$

Estimating the Policy Gradient

- Estimating the policy gradient methods
  - Finite Difference Methods: These methods approximate the gradient by perturbing the policy parameters and observing the change in the expected return.
  - Score Function Methods: Score function methods, directly estimate the gradient of the expected return using samples obtained from interactions with the environment.

Estimating the Policy Gradient

- Estimating the policy gradient methods
  - Likelihood Ratio Methods: Likelihood ratio methods, further improve the efficiency of gradient estimation by subtracting a baseline function from the returns before computing the gradient.
  - Actor-Critic Methods: Actor-critic methods combine policy gradient estimation with value function estimation.

Estimating the Policy Gradient

- Estimating the policy gradient methods
  - Natural Policy Gradient: Natural policy gradient methods incorporate the geometry of the parameter space into gradient estimation to improve convergence properties and stability.

Actor-Critic Algorithm

- The Actor-Critic algorithm combines elements of both value-based and policy-based methods.
- It consists of two main components: the actor and the critic:
  - **Actor**: The actor is responsible for learning the policy
  - **Critic**: The critic is responsible for evaluating the policy.
- The Actor-Critic algorithm operates by iteratively updating the actor and critic networks based on observed experiences in the environment.

Actor-Critic Algorithm

- Actor- Critic Algorithm:
  - Step 1- Initialization: Initialize the actor and critic networks with random parameters.
  - Step 2- Interact with Environment: Sample trajectories by following the current policy in the environment.
  - Step 3- Compute Returns: Compute the returns for each time step in the trajectory.
  - Step 4- Update Critic: Use the returns to update the parameters of the critic network

Actor-Critic Algorithm

- Actor- Critic Algorithm:
  - Step 5- Compute Advantages: Compute advantages for each time step by subtracting the estimated value from the observed return.
  - Step 6- Update Actor: Update the parameters of the actor network using policy gradients
  - Step 7- Repeat: Repeat steps 2-6 for multiple episodes or until convergence criteria are met.

Actor-Critic Algorithm

- The Actor-Critic algorithm in a simple grid world environment

```python
# 3.11 Actor-Critic Algorithm- HoaDNt@fe.edu.vn
import numpy as np

class GridWorld:
    def __init__(self):
        self.grid_size = (3, 3)
        self.num_actions = 4  # Up, Down, Left, Right
        self.start_state = (0, 0)
        self.goal_state = (2, 2)

    def step(self, state, action):
        # Define the dynamics of the environment
        row, col = state
        if action == 0:  # Up
            row = max(0, row - 1)
        elif action == 1:  # Down
            row = min(self.grid_size[0] - 1, row + 1)
        elif action == 2:  # Left
            col = max(0, col - 1)
        elif action == 3:  # Right
            col = min(self.grid_size[1] - 1, col + 1)
        next_state = (row, col)
        reward = 0
        if next_state == self.goal_state:
            reward = 1  # Reward of +1 upon reaching the goal state
        return next_state, reward
```

Actor-Critic Algorithm

```
class ActorCritic:
    def __init__(self, num_actions, alpha_actor, alpha_critic, gamma):
        self.num_actions = num_actions
        self.alpha_actor = alpha_actor
        self.alpha_critic = alpha_critic
        self.gamma = gamma
        self.actor_params = np.zeros((3, 3, num_actions))  # Tabular actor parameters
        self.critic_values = np.zeros((3, 3))  # Tabular critic values

    def select_action(self, state):
        # Select action probabilistically based on actor parameters
        action_probs = self.softmax(self.actor_params[state])
        action = np.random.choice(self.num_actions, p=action_probs)
        return action

    def update(self, state, action, reward, next_state):
        # Compute TD error (advantage)
        td_error = reward + self.gamma * self.critic_values[next_state] - self.critic_values[state]

        # Update critic values
        self.critic_values[state] += self.alpha_critic * td_error

        # Update actor parameters
        self.actor_params[state][action] += self.alpha_actor * td_error

    def softmax(self, x):
        e_x = np.exp(x - np.max(x))
        return e_x / e_x.sum(axis=0)
```

Actor-Critic Algorithm

```python
# Create a grid world environment
grid_world = GridWorld()

# Create an Actor-Critic agent
num_actions = 4  # Up, Down, Left, Right
alpha_actor = 0.1
alpha_critic = 0.1
gamma = 0.9
actor_critic_agent = ActorCritic(num_actions, alpha_actor, alpha_critic, gamma)

# Train the Actor-Critic agent
num_episodes = 1000
for _ in range(num_episodes):
    state = grid_world.start_state
    while state != grid_world.goal_state:
        action = actor_critic_agent.select_action(state)
        next_state, reward = grid_world.step(state, action)
        actor_critic_agent.update(state, action, reward, next_state)
        state = next_state

# Evaluate the learned policy
total_reward = 0
state = grid_world.start_state
while state != grid_world.goal_state:
    action = actor_critic_agent.select_action(state)
    next_state, reward = grid_world.step(state, action)
    total_reward += reward
    state = next_state

print("Total reward obtained by learned policy:", total_reward)
```

FPT Education
FPT UNIVERSITY

Actor-Critic Algorithm

- Run that code and show a total reward obtained by learned policy

Summary

- ❑ Derive a sample-based estimate for the gradient
- ❑ Understand the Actor-Critic algorithm

Q & A