FPT Education
FPT UNIVERSITY

Temporal Difference for Control

Objectives

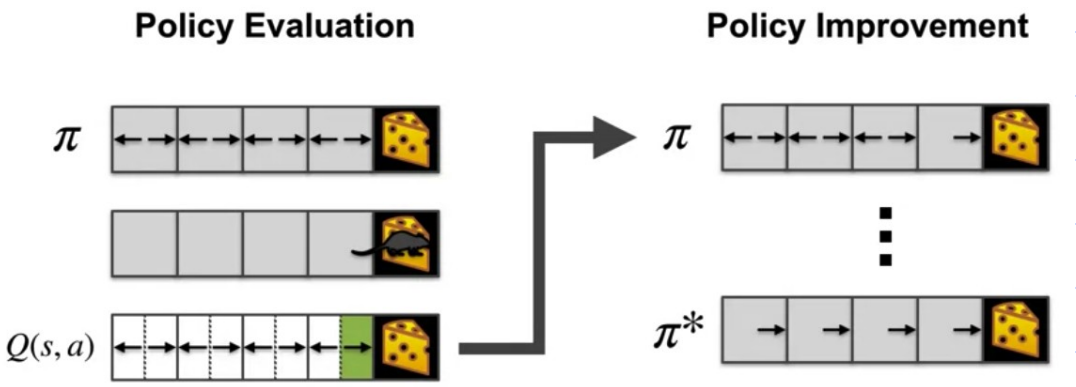
- Explain how generalized policy iteration can be used with TD to find improved policies
- Describe the Sarsa Control algorithm

GPI with Temporal Difference

- Generalized Policy Iteration (GPI) is a framework in reinforcement learning that combines policy evaluation and policy improvement in an iterative manner.
- It involves continuously alternating between evaluating the current policy and improving it based on the estimated value functions.
- Temporal Difference (TD) learning can be integrated into the GPI framework to find improved policies

GPI with Temporal Difference

- Monte Carlo does not perform a full policy evaluation step before improvement.
- Rather, it evaluates and improves after each episode.
- We could improve the policy after just one policy evaluati

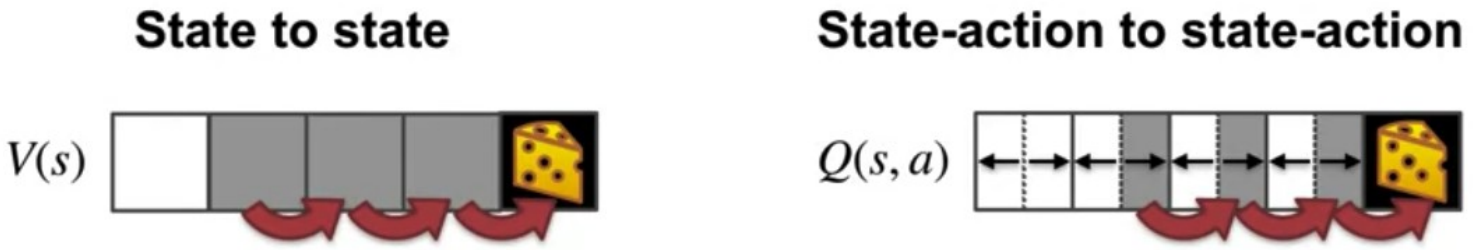


SARSA Algorithm

- SARSA (State-Action-Reward-State-Action) is a temporal difference learning algorithm commonly used in reinforcement learning.
  - SARSA is an on-policy algorithm, meaning it learns the value function and policy while following the same policy being evaluated.
  - SARSA can be used for prediction, which involves estimating the value function Q(s,a), representing the expected cumulative reward starting from state s and taking action a, under a given policy.

SARSA Algorithm

- From state action pair to state action pair and learn the value of each pair



SARSA Algorithm steps

- Initialization:
  - Initialize the state-action value function Q(s,a) arbitrarily for all state-action pairs.
- Episodic Interaction:
  - Interact with the environment by following a policy.
  - Generate episodes of interaction, where each episode consists of a sequence of state-action-reward-state-action transitions.

SARSA Algorithm

- Action Selection:
  - At each time step t, select an action at in the current state st based on the policy.
  - This could be an ε-greedy policy, where with probability ε a random action is chosen, and with probability 1−ε the action with the highest value estimate Q(s t ,a) is selected.

SARSA Algorithm

- Update Rule: After selecting an action a t and observing the reward r t+1and the next state st+1, update the value estimate for the current state-action pair Q(s t ,at) using the SARSA update rul $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right)$
- where:
  - α is the learning rate, controlling the size of the updates
  - γ is the discount factor, representing the importance of future rewards relative to immediate rewards.
  - a t+1is the next action chosen in states t+1according to

SARSA Algorithm

- Steps
  - Set the initial state s.
  - Choose the initial action a using an epsilon-greedy policy based on the current Q values.
  - Take the action a and observe the reward r and the next state s’.
  - Choose the next action a’ using an epsilon-greedy policy based on the updated Q values.
  - Update the action-value estimate for the current state-action pair using the SARSA update rule:

Q(s, a) = Q(s, a) + alpha * (r + gamma * Q(s’, a’) – Q(s, a))

Repeat steps 4-7 until the episode ends.

SARSA Algorithm Example code

- Step 1: Importing the required libraries

```
import numpy as np
import gym
```

- Step 2: Building the environment: using the 'FrozenLake-v0' environment which is preloaded into gym

```
#Building the environment
env = gym.make('FrozenLake-v0')
```

SARSA Algorithm Example code

- Step 3: Initializing different parameters

```
#Defining the different parameters
epsilon = 0.9
total_episodes = 10000
max_steps = 100
alpha = 0.85
gamma = 0.95

#Initializing the Q-matrix
Q = np.zeros((env.observation_space.n, env.action_space.n))
```

SARSA Algorithm Example code

- Step 4: Defining utility functions to be used in the learning process

```
#Function to choose the next action
def choose_action(state):
    action=0
    if np.random.uniform(0, 1) < epsilon:
        action = env.action_space.sample()
    else:
        action = np.argmax(Q[state, :])
    return action

#Function to learn the Q-value
def update(state, state2, reward, action, action2):
    predict = Q[state, action]
    target = reward + gamma * Q[state2, action2]
    Q[state, action] = Q[state, action] + alpha * (target - predict)
```

SARSA Algorithm Example code

- Step 5: Training the learning agent

```
#Initializing the reward
reward=0

# Starting the SARSA learning
for episode in range(total_episodes):
    t = 0
    state1 = env.reset()
    action1 = choose_action(state1)

    while t < max_steps:
        #Visualizing the training
        env.render()

        #Getting the next state
        state2, reward, done, info = env.step(action1)

        #Choosing the next action
        action2 = choose_action(state2)
```

SARSA Algorithm Example code

- Step 5: Training the learning agent

```
#Learning the Q-value
update(state1, state2, reward, action1, action2)

state1 = state2
action1 = action2

#Updating the respective vaLues
t += 1
reward += 1

#If at the end of learning process
if done:
    break
```

SARSA Algorithm Example code

- Step 6: Evaluating the performance

```
#Evaluating the performance
print ("Performance : ", reward/total_episodes)

#Visualizing the Q-matrix
print(Q)
```

```
Performace :  0.0001
[[4.03497050e-04 3.44940226e-03 2.95271581e-04 1.90817997e-04]
 [1.15418001e-05 7.41311564e-04 1.03167884e-03 1.66006591e-03]
 [5.58624322e-04 1.43586642e-03 9.31204292e-04 5.41248854e-04]
 [2.69757315e-03 1.73146586e-04 4.46039395e-06 2.30746180e-04]
 [3.26589484e-04 1.27060872e-03 1.06355375e-06 4.59986098e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.83146552e-01 4.59926829e-02 2.31092412e-04 1.06945999e-03]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [1.90161780e-03 2.23215112e-04 6.43823638e-03 2.00008656e-03]
 [1.38638998e-01 1.08892484e-01 1.20888168e-02 3.61722497e-03]
 [8.35442395e-02 1.05179959e-02 7.20923321e-03 2.36916944e-04]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [3.99223624e-03 9.86266131e-02 5.78735337e-01 5.91825955e-01]
 [1.22916541e-01 2.07281906e-02 2.22069056e-01 5.53015119e-01]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

Summary

- Explain how generalized policy iteration can be used with TD to find improved policies
- Describe the Sarsa Control algorithm

FPT Education
FPT UNIVERSITY

Q & A