FPT Education
FPT UNIVERSITY

Episodic Sarsa with Function Approximation

Objectives

- Understand how to construct action-dependent features for approximate action values
- Explain the update for Episodic Sarsa with function approximation
- Explain the update for expected Sarsa with function approximation.
- Explain the update for Q-learning with function approximation.

Episodic Sarsa with Function Approximation

- The State-values to action-values
  - The value function approximation has two components, a weight vector and a feature vector.
  - In a given state, the value estimate is the dot product between these two components.

$$v_\pi(s) \approx \hat{v}(s, \mathbf{w}) \doteq \mathbf{w}^T \mathbf{x}(s)$$

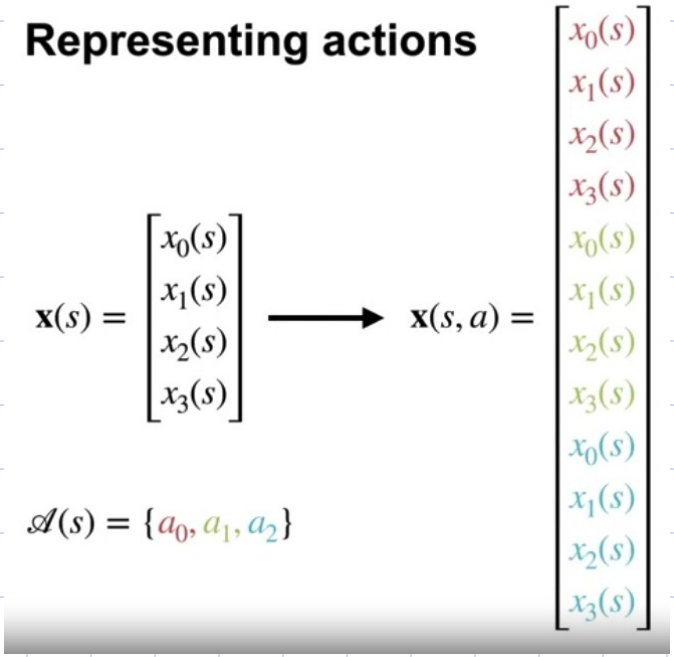
Episodic Sarsa with Function Approximation

- The State-values to action-values
  - To move from TD to Sarsa, we need action value functions.
  - The feature representation has to represent actions as

$$q_\pi(s,a) \approx \hat{q}(s,a,\mathbf{w}) \doteq \mathbf{w}^T\mathbf{x}(s,a)$$

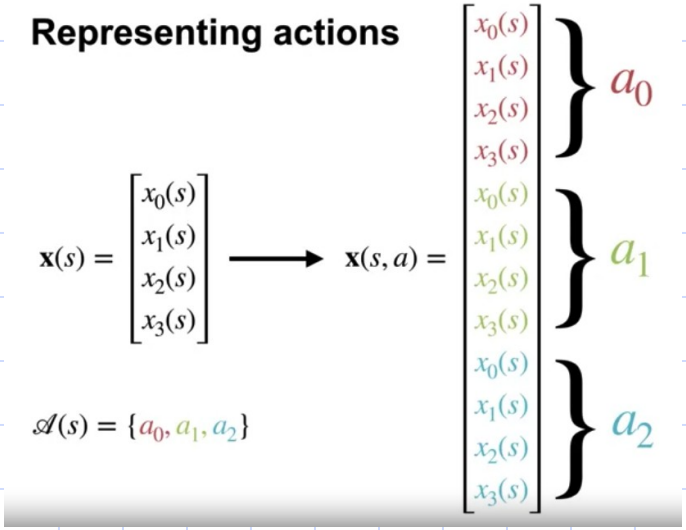
Episodic Sarsa with Function Approximation

- Example:
  - There are four features and three actions.
  - The four features represent the state you are in.
  - But we want to learn a function of both states and actions.
  - We can do this by repeating the four features for each action.



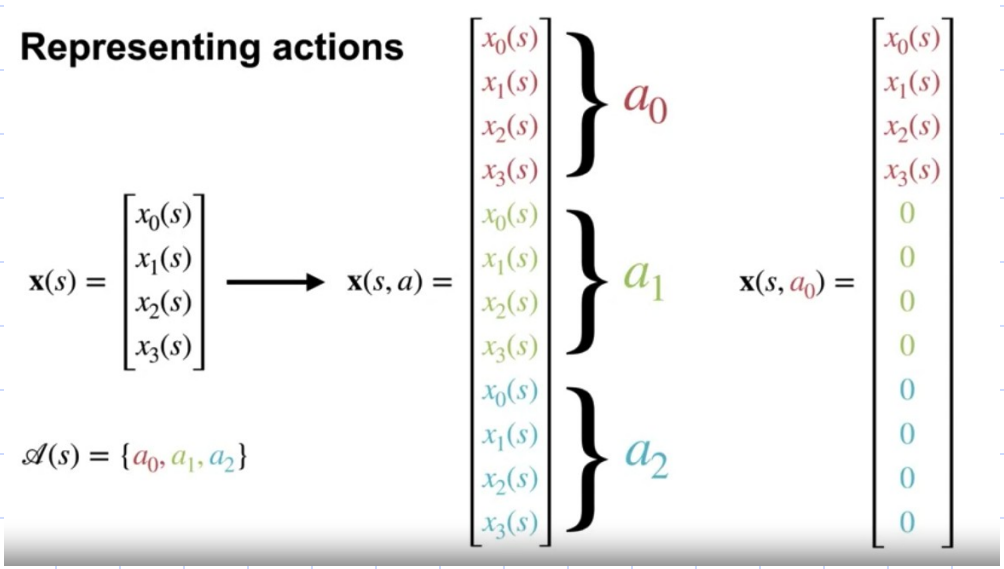
Episodic Sarsa with Function Approximation

- Example:
  - The feature vector has 12 components.
  - Here, each segment of four features corresponds to a separate action.
  - We call this feature representation stacked because the weights for each action are stacked on top of each other.



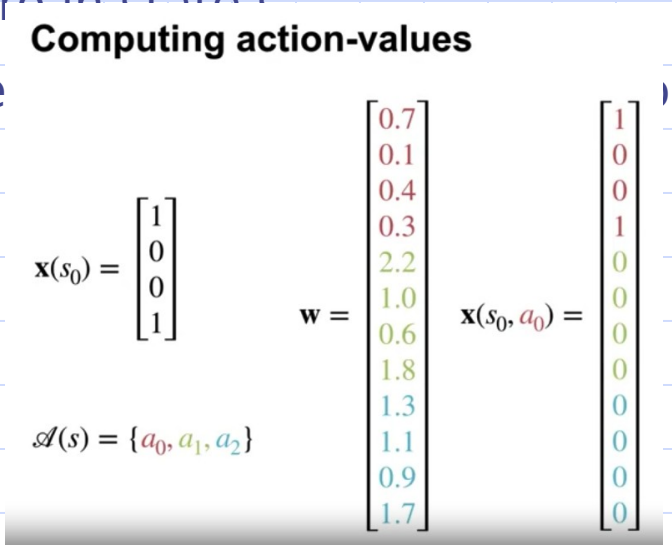
Episodic Sarsa with Function Approximation

- Example:
  - Only the features for the specified action will be active, while though for the other actions will be set to 0



Episodic Sarsa with Function Approximation

- Example: calculate the action values from a given state
  - There are four features and three actions.
  - The weight factor looks like figure.
  - With stacked features, we get the following feature vector for action zero in state s
  - We 0 out the ons.



Episodic Sarsa with Function Approximation

- Example: calculate the action values from a given state
  - We extract the segment of the weight vector correspo

Computing action-values

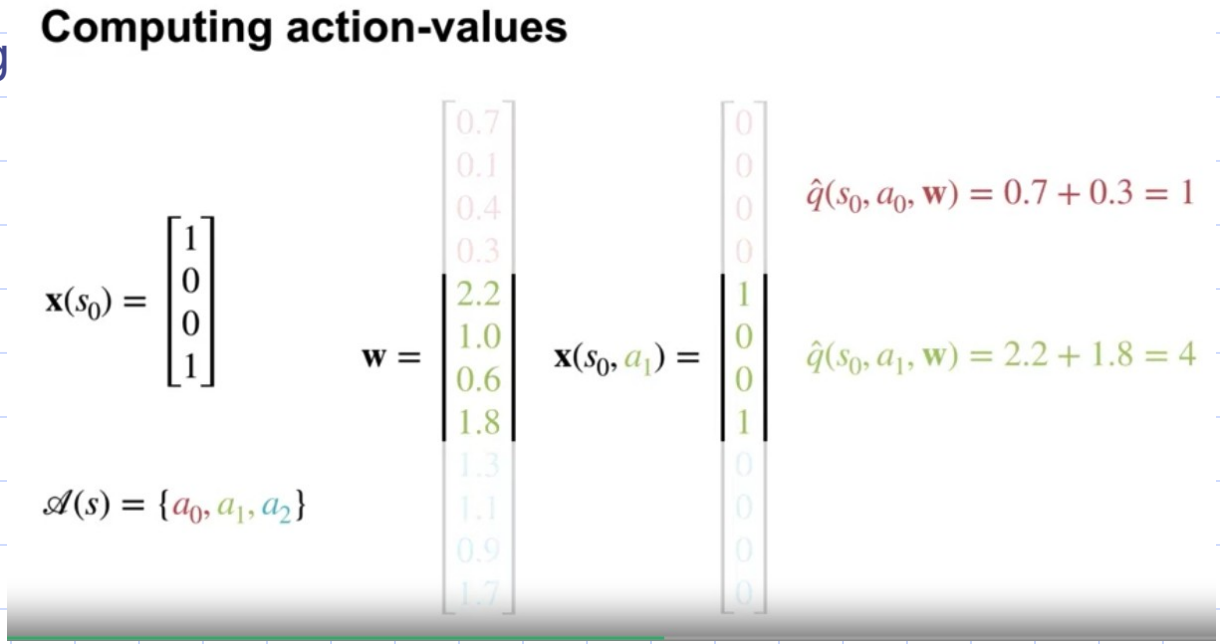
$$\mathbf{x}(s_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.4 \\ 0.3 \\ 2.2 \\ 1.0 \\ 0.6 \\ 1.8 \\ 1.3 \\ 1.1 \\ 0.9 \\ 1.7 \end{bmatrix} \qquad \mathbf{x}(s_0, a_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

$$\hat{q}(s_0, a_0, \mathbf{w}) = 0.7 + 0.3 = 1$$

Episodic Sarsa with Function Approximation

- Example: calculate the action values from a given state
  - The action values are then the dot products between each seg



Episodic Sarsa with Function Approximation

- Example: calculate the action values from a given state

Computing action-values

$$\mathbf{x}(s_0) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} 0.7 \\ 0.1 \\ 0.4 \\ 0.3 \\ 2.2 \\ 1.0 \\ 0.6 \\ 1.8 \\ 1.3 \\ 1.1 \\ 0.9 \\ 1.7 \end{bmatrix} \qquad \mathbf{x}(s_0, a_2) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathcal{A}(s) = \{a_0, a_1, a_2\}$$

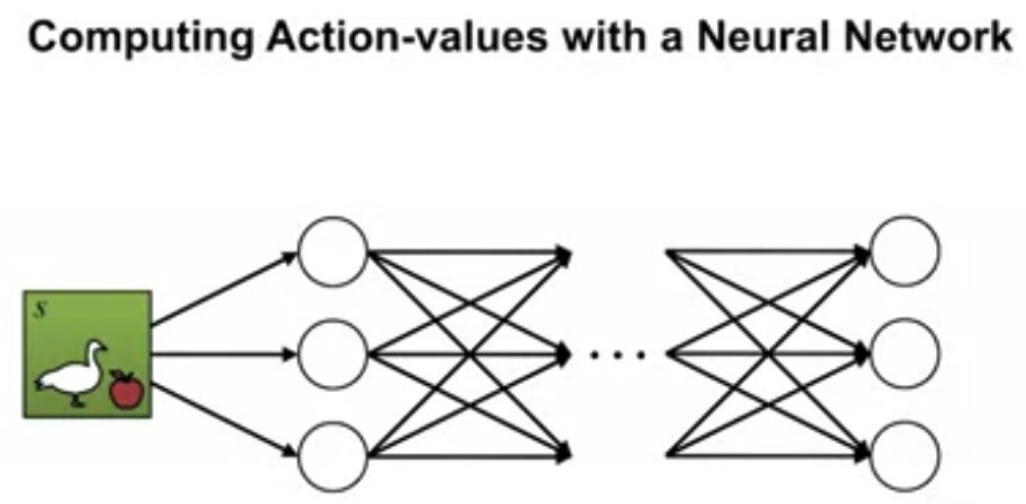
$$\hat{q}(s_0, a_0, \mathbf{w}) = 0.7 + 0.3 = 1$$

$$\hat{q}(s_0, a_1, \mathbf{w}) = 2.2 + 1.8 = 4$$

$$\hat{q}(s_0, a_2, \mathbf{w}) = 1.3 + 1.7 = 3$$

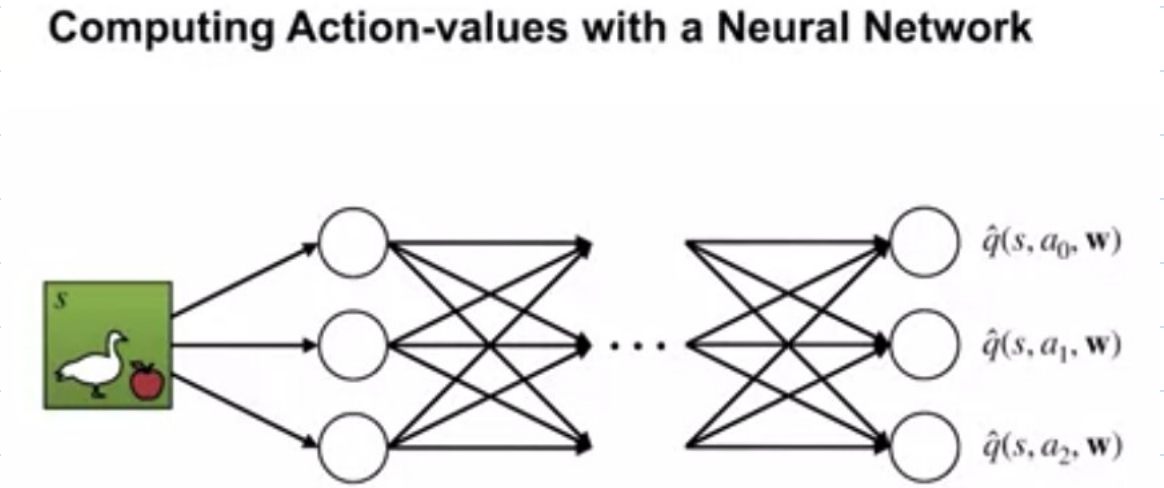
Episodic Sarsa with Function Approximation

- Computing action-values with a Neural Network
  - The common way to represent action values with a neural network is to generate multiple outputs, one for each action value.



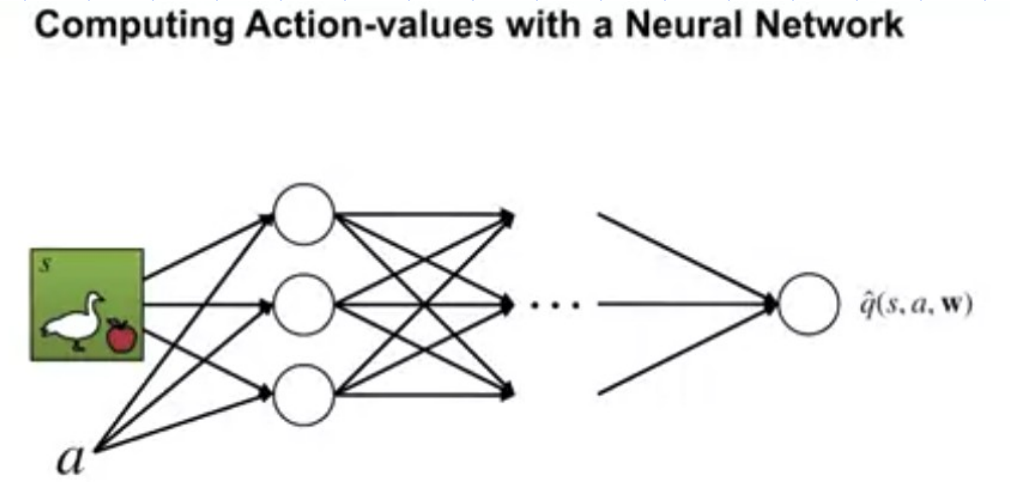
Episodic Sarsa with Function Approximation

- Computing action-values with a Neural Network
  - The neural network inputs the state and the last hidden layer produces the state features.
  - Each action value is computed from an independent set of weights using those state features.



Episodic Sarsa with Function Approximation

- Computing action-values with a Neural Network
  - We would input both the state and the action to the network. There would only be one output.
  - The approximate action value for that state and action.
  - We can do something similar with tile coating by passing both the state and action as input



Episodic Sarsa with Function Approximation

- Sarsa for control of function approximation

Episodic Semi-gradient Sarsa for Estimating $\hat{q} \approx q_*$

Input: a differentiable action-value function parameterization $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$
Algorithm parameters: step size $\alpha > 0$, small $\varepsilon > 0$
Initialize value-function weights $\mathbf{w} \in \mathbb{R}^d$ arbitrarily (e.g., $\mathbf{w} = \mathbf{0}$)

Loop for each episode:
  $S, A \leftarrow$ initial state and action of episode (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    If S' is terminal:
      $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
      Go to next episode
    Choose A' as a function of $\hat{q}(S', \cdot, \mathbf{w})$ (e.g., ε-greedy)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha\big[R + \gamma\hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})\big]\nabla\hat{q}(S, A, \mathbf{w})$
    $S \leftarrow S'$
    $A \leftarrow A'$

Episodic Sarsa with Function Approximation

- From Sarsa to Expected Sarsa
  - Sarsa: Sarsa's update target includes the action value for the next state in action.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\right)$$

  - Expected Sarsa: Sum over the action values weighted by their probability under the target policy.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \sum_{a'} \pi(a' \mid S_{t+1}) Q(S_{t+1}, a') - Q(S_t, A_t)\right)$$

Expected Sarsa with Function Approximation

- The update for Sarsa with function approximation.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\big(R_{t+1} + \gamma\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\big)\nabla\hat{q}(S_t, A_t, \mathbf{w})$$

- The Expected Sarsa:
  - We compute the action values from our weight vector for every action in the next state. Then we compute this expectation under the target policy.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\Big(R_{t+1} + \gamma\sum_{a'}\pi(a'|S_{t+1})\hat{q}(S_{t+1}, a', \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\Big)\nabla\hat{q}(S_t, A_t, \mathbf{w})$$

$$q_\pi(s, a) \approx \hat{q}(s, a, \mathbf{w})$$

Expected Sarsa to Q-Learning

- The Expected Sarsa:
  - We compute the action values from our weight vector for every action in the next state. Then we compute this expectation under the target policy.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\Big(R_{t+1} + \gamma \sum_{a'} \pi(a' \mid S_{t+1}) \hat{q}(S_{t+1}, a', \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\Big) \nabla \hat{q}(S_t, A_t, \mathbf{w})$$

- Q-Learning: use the max in place of the ex

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\Big(R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a', \mathbf{w}) - \hat{q}(S_t, A_t, \mathbf{w})\Big) \nabla \hat{q}(S_t, A_t, \mathbf{w})$$

$$q_\pi(s, a) \approx \hat{q}(s, a, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s, a)$$

Sarsa Example

- The demonstrates how SARSA can be used to learn the optimal policy for a grid world environment by updaating the Q-values based on observed rewards and subsequent actions taken.

Sarsa Example

```python
# 3.6 Implement SARSA- HoaDNT@fe.edu.vn
import numpy as np

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

    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

```

Sarsa Example

```
27
28 def epsilon_greedy_policy(Q, state, epsilon):
29     if np.random.rand() < epsilon:
30         # Explore: choose a random action
31         action = np.random.randint(len(Q[state]))
32     else:
33         # Exploit: choose the action with the highest Q-value
34         action = np.argmax(Q[state])
35     return action
36
37 def sarsa(grid_world, num_episodes, alpha, gamma, epsilon):
38     # Initialize Q-value function
39     Q = np.zeros((grid_world.grid_size[0], grid_world.grid_size[1], grid_world.num_actions))
40
41     for _ in range(num_episodes):
42         state = grid_world.start_state
43         action = epsilon_greedy_policy(Q, state, epsilon)
44         while state != grid_world.goal_state:
45             next_state, reward = grid_world.step(state, action)
46             next_action = epsilon_greedy_policy(Q, next_state, epsilon)
47             # SARSA update rule
48             Q[state][action] += alpha * (reward + gamma * Q[next_state][next_action] - Q[state][action])
49             state = next_state
50             action = next_action
51
52     return Q
53
```

Sarsa Example

```
53
54 # Create a grid world environment
55 grid_world = GridWorld()
56
57 # Set hyperparameters
58 num_episodes = 1000
59 alpha = 0.1  # Learning rate
60 gamma = 0.9  # Discount factor
61 epsilon = 0.1  # Epsilon-greedy exploration parameter
62
63 # Run SARSA algorithm to learn the optimal policy
64 Q = sarsa(grid_world, num_episodes, alpha, gamma, epsilon)
65
66 # Print the learned Q-value function
67 print("Learned Q-value Function:")
68 print(Q)
69
```

Sarsa Example

```
Learned Q-value Function:
[[[0.40051576 0.23408297 0.43206549 0.51117513]
  [0.41452949 0.33254619 0.36193039 0.59456359]
  [0.51943338 0.67683376 0.47848312 0.41728476]
  [0.08213813 0.         0.57986879 0.08553801]]

 [[0.39799241 0.         0.         0.10004665]
  [0.49882321 0.         0.         0.        ]
  [0.50018563 0.4660517  0.40355613 0.78770142]
  [0.41813902 0.87075818 0.59889402 0.66730315]]

 [[0.         0.         0.         0.        ]
  [0.         0.         0.         0.        ]
  [0.62445957 0.         0.         0.14965436]
  [0.64245207 1.         0.55682119 0.70921377]]

 [[0.         0.         0.         0.        ]
  [0.         0.         0.         0.        ]
  [0.         0.         0.         0.        ]
  [0.         0.         0.         0.        ]]]
```

Summary

- Understand how to construct action-dependent features for approximate action values
- Explain the update for Episodic Sarsa with function approximation
- Explain the update for expected Sarsa with function approximation.
- Explain the update for Q-learning with function approximation.

Q & A