7. Linear Q-Learning

Extension Note: Project 5 due date has been extended by 1 **more** day to **September 6 23:59UTC** .

In this tab, you will implement the Q-learning algorithm with linear function approximation.

Recall the linear approximation we chose.

$$Q(s,c,\theta) = \phi(s,c)^T\theta$$

with

$$\phi(s,c) = \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \psi_R(s) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}$$

Now, define $\hat{\theta}_i$ for i in range $1, d_C$ so that:

$$\theta = \begin{bmatrix} \hat{\theta}_1 \\ \vdots \\ \hat{\theta}_i \\ \vdots \\ \hat{\theta}_{d_C} \end{bmatrix}$$

With this notation, we get:

$$Q(s,c,\theta) = \psi_R(s)^T\hat{\theta}_c$$

In practice, we can implement $\hat{\theta}$ as a 2D array, so that

$$\begin{bmatrix} Q(s,1,\theta) \\ \vdots \\ Q(s,d_C,\theta) \end{bmatrix} = \begin{bmatrix} \hat{\theta}_1^T \\ \vdots \\ \hat{\theta}_{d_C}^T \end{bmatrix} \cdot \psi_R(s)$$

Epsilon-greedy exploration

1.0/1 point (graded)

Now you will write a function `epsilon\_ greedy` that implements the ε-greedy exploration policy using the current Q-function.

Hint: You can access $Q(s, c, \theta)$ using `q_value = (theta @ state_vector)[tuple2index(action_index, object_index)]`

Available Functions: You have access to the NumPy python library as `np` and functions `tuple2index` and `index2tuple`. Your code should also use constants `NUM_ACTIONS` and `NUM_OBJECTS`

```
    Args:
        state_vector (np.ndarray): extracted vector representation
        theta (np.ndarray): current weight matrix
        epsilon (float): the probability of choosing a random command

    Returns:
        (int, int): the indices describing the action/object to take
    """
    if np.random.random() > epsilon:
        q_state = theta @ state_vector
        action_index, object_index = index2tuple(q_state.argmax())
    else:
        action_index, object_index = np.random.randint(NUM_ACTIONS), np.random.randint(NUM_OBJECTS)
    return (action_index, object_index)

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def epsilon_greedy(state_vector, theta, epsilon):
    """Returns an action selected by an epsilon-greedy exploration policy

    Args:
        state_vector (np.ndarray): extracted vector representation
        theta (np.ndarray): current weight matrix
        epsilon (float): the probability of choosing a random command

    Returns:
        (int, int): the indices describing the action/object to take
    """
    coin = np.random.random_sample()
    if coin < epsilon:
        action_index = np.random.randint(NUM_ACTIONS)
        object_index = np.random.randint(NUM_OBJECTS)
    else:
        q_values = theta @ state_vector
        index = np.argmax(q_values)
        action_index, object_index = index2tuple(index)
    return (action_index, object_index)
```

Test results

See full output

CORRECT

See full output

Submit

You have used 4 of 25 attempts

Answers are displayed within the problem

Linear Q-learning

1.0/1 point (graded)

Write a function `linear_q_learning` that updates the theta weight matrix, given the transition date $(s, a, R(s, a), s')$.

Reminder: You should implement this function locally first. You should test this function along with the next one and make sure you achieve reasonable performance

Hint: You can access $Q(s, a, \theta)$ using `q_value = (theta @ state_vector)[tuple2index(action_index, object_index)]`

Available Functions: You have access to the NumPy python library as `np`. You should also use constants `ALPHA` and `GAMMA` in your code

```
    Returns:
```

```
        None
    """
    if not terminal:
        v_func = np.max(theta @ next_state_vector)
    else:
        v_func = 0

    target_y = reward + GAMMA * v_func
    idx = tuple2index(action_index, object_index)
    q_value = (theta @ current_state_vector)[idx]
    theta[idx] += ALPHA * current_state_vector * (target_y - q_value)

    return None

```

Press ESC then TAB or click outside of the code editor to exit

Correct

```
def linear_q_learning(theta, current_state_vector, action_index, object_index,
                      reward, next_state_vector, terminal):
    """Update theta for a given transition

    Args:
        theta (np.ndarray): current weight matrix
        current_state_vector (np.ndarray): vector representation of current state
        action_index (int): index of the current action
        object_index (int): index of the current object
        reward (float): the immediate reward the agent recieves from playing current command
        next_state_vector (np.ndarray): vector representation of next state
        terminal (bool): True if this epsiode is over

    Returns:
        None
    """
    q_values_next = theta @ next_state_vector
    maxq_next = np.max(q_values_next)

    q_values = theta @ current_state_vector
    cur_index = tuple2index(action_index, object_index)
    q_value_cur = q_values[cur_index]

    target = reward + GAMMA * maxq_next * (1 - terminal)

    theta[cur_index] = theta[cur_index] + ALPHA * (
        target - q_value_cur) * current_state_vector
```

Test results

See full output

CORRECT

See full output

Submit You have used 3 of 25 attempts

Answers are displayed within the problem

Evaluate linear Q-learning on Home World game

1/1 point (graded)

Adapt your `run_episode` function to call `linear_Q_learning` and evaluate your performance using hyperparameters:

Set `NUM_RUNS` $= 5$, `NUM_EPIS_TRAIN` $= 25$, `NUM_EPIS_TEST` $= 50$, $\gamma = 0.5$, `TRAINING_EP` $= 0.5$, `TESTING_EP` $= 0.05$ and the learning rate $\alpha = 0.001$.

Please enter the *average episodic rewards* of your Q-learning algorithm when it converges.

0.4 ✔ **Answer:** 0.37

Submit You have used 2 of 6 attempts

Answers are displayed within the problem

Discussion

Topic: Unit 5 Reinforcement Learning (2 weeks) :Project 5: Text-Based Game / 7. Linear Q-Learning

Show Discussion