Day 2. Tabular MDPs

NPEX Reinforcement Learning

NPEX 2020

Jaeuk Shin

CORE
Control + Optimization Research Lab

MDP - Review

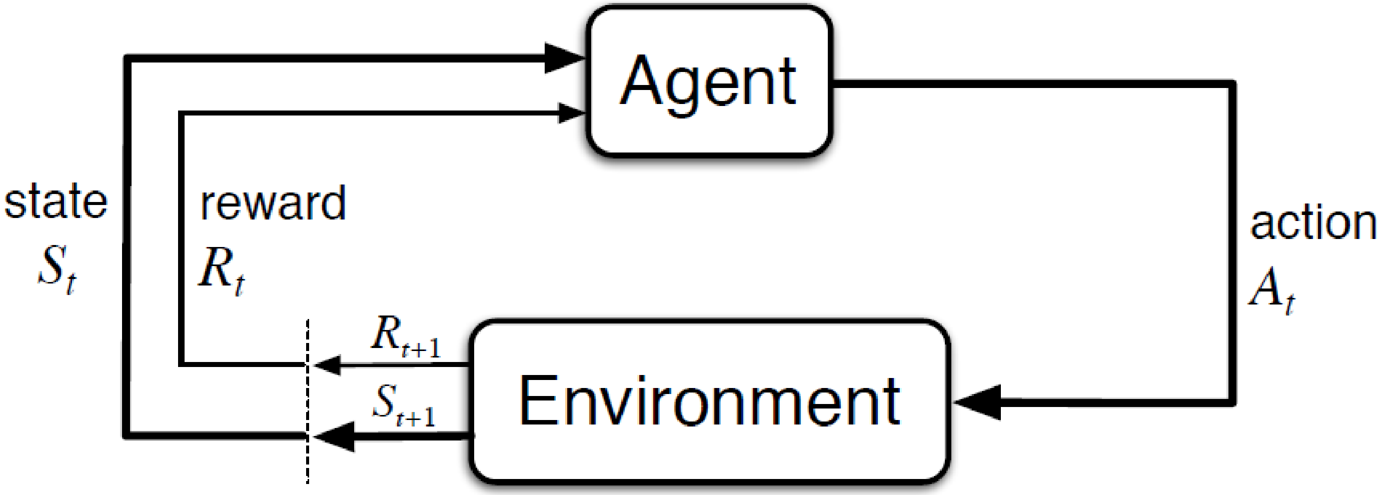
$$\mathcal{S} = \{s_0, \cdots s_{n-1}\}\text{: state space}$$

$$\mathcal{A} = \{a_0, \cdots a_{m-1}\}\text{: action space}$$

$$p(s'|s,a)\text{: transition probability}$$

$$r(s,a)\text{: reward function}$$

$$\gamma\text{: discount rate}$$

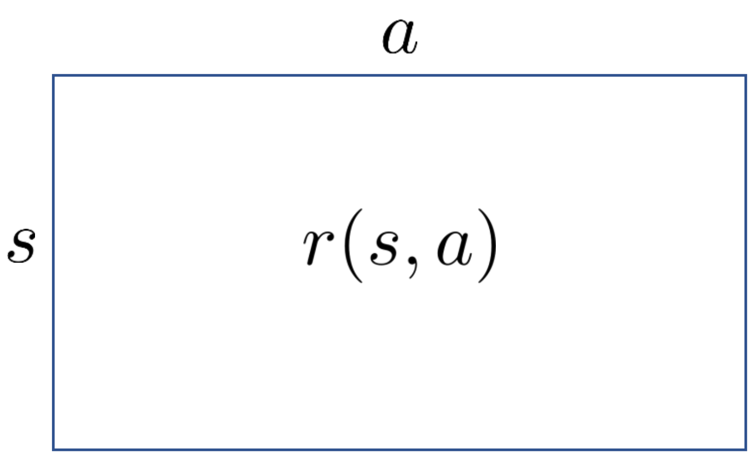


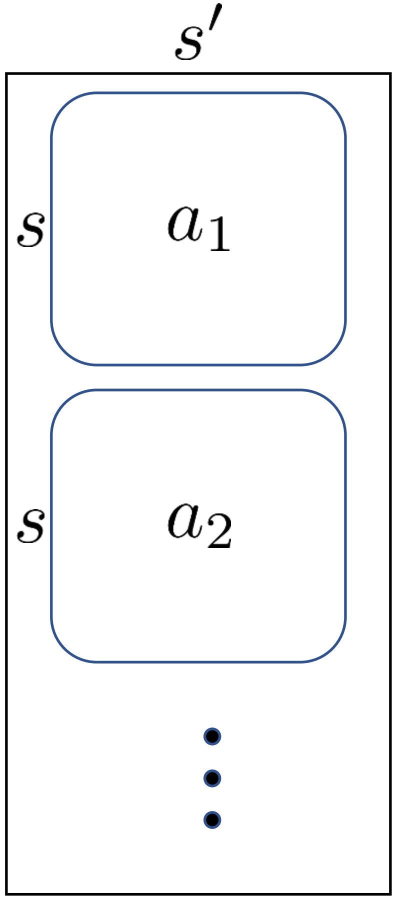
MDP - Review

How to represent these data?

transition probability $p(s'|s,a)$: matrix P of size $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$:

reward function $r(s,a)$: matrix R of size $|\mathcal{S}| \times |\mathcal{A}|$:





MDP - Review

$$\mathcal{S} = \{s_0, s_1\}, \quad \mathcal{A} = \{a_0, a_1\},$$
$$r(s_0, a_0) = -2, \quad r(s_0, a_0) = -0.5,$$
$$r(s_1, a_0) = -1, \quad r(s_1, a_1) = -3.0,$$
$$p(s_0|s_0, a_0) = 0.75,$$
$$p(s_0 \; |s_1, a_0) = 0.75,$$
$$p(s_0 \; |s_0, a_1) = 0.25,$$
$$p(s_0 \; |s_1, a_1) = 0.25.$$

```
R = np.array([[-2.0, -0.5],
              [-1.0, -3.0]])

P = np.array([[0.75, 0.25],
              [0.75, 0.25],
              [0.25, 0.75],
              [0.25, 0.75]])
```

Solving Tabular MDPs – Value Iteration

Review : **Bellman operator** $\mathcal{T} : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{S}}$ is given by

$$(\mathcal{T}v)(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a)v(s') \right)$$

Given a vector v of size $n \times 1$,

Step 1. compute $r(s,a) + \gamma \sum_{s'} p(s'|s,a)v(s')$,

Step 2. and then take $\max_a$.

Solving Tabular MDPs – Value Iteration

Step 1. compute $r(s,a) + \gamma \sum_{s'} p(s'|s,a)v(s')$,

```python
def q_ftn(P, R, gamma, v):
    """
    given v, get corresponding q
    """
    return R + gamma * np.reshape(np.matmul(P, v), newshape=R.shape, order='F')
```

Shape of $q(s,a)$?

Step 2. and then take $\max_a$.

```python
def bellman_update(P, R, gamma, v):
    """
    implementation of one-step Bellman update
    return : vector of shape (|S|, 1) which corresponds to Tv, where T is Bellman operator
    """

    q = q_ftn(P, R, gamma, v)
    v_next = np.max(q, axis=1, keepdims=True)    # computation of Bellman operator Tv

    return v_next
```

CORE
Control + Optimization Research Lab

Solving Tabular MDPs – Value Iteration

$$\pi(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s'} p(s'|s,a) v(s') \right)$$

```
def greedy(P, R, gamma, v):
    """
    construct greedy policy by pi(s) = argmax_a q(s, a)
    """
    q = q_ftn(P, R, gamma, v)
    pi = np.argmax(q, axis=1)

    return pi
```

Combining all of these, we have...

Solving Tabular MDPs – Value Iteration

```python
def VI(P, R, gamma):
    """
    implementation of value iteration
    """
    EPS = 1e-6
    nS, nA = R.shape
    # initialize v
    v = np.zeros(shape=(nS, 1), dtype=np.float)

    while True:
        v_next = bellman_update(P, R, gamma, v)
        if np.linalg.norm(v_next - v, ord=np.inf) < EPS
            break
        v = v_next

    pi = greedy(P, R, gamma, v)

    return v, pi
```

(terminal condition)

$$\max_s |v(s) - (\mathcal{T}v)(s)| \leq \epsilon$$

Solving Tabular MDPs – Policy Iteration

Review : any policy π satisfies

$$v^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) v^\pi(s').$$

Step 1. compute v^π by solving the above equation (Policy Evaluation)

Step 2. determine π_{next} greedily (Policy Improvement):

$$\pi_{\text{next}}(s) = \arg\max_a \left(r(s, a) + \gamma \sum_{s'} p(s'|s, a) v^\pi(s') \right)$$

Solving Tabular MDPs – Policy Iteration

Step 1. compute v^π by solving the above equation (Policy Evaluation)

```
def induced_dynamic(nS, P, R, pi):
    """
    given policy pi, compute induced dynamic P^pi & R^pi
    """
    S = range(nS)
    P_pi = np.array([P[pi[s] * nS + s, :] for s in S])
    R_pi = np.array([[R[s, pi[s]]] for s in S])

    return P_pi, R_pi
```

```
def eval_policy(nS, P, R, gamma, pi):
    """
    policy evaluation
    """
    P_pi, R_pi = induced_dynamic(nS, P, R, pi)

    Id = np.identity(nS)

    # discounted reward problem
    v_pi = np.linalg.solve(Id - gamma * P_pi, R_pi)
    return v_pi
```

$$P^\pi = \begin{pmatrix} p(0|0,\pi(0)) & \cdots & p(n-1|0,\pi(0)) \\ \vdots & \vdots & \vdots \\ p(0|n-1,\pi(n-1)) & \cdots & p(n-1|n-1,\pi(n-1)) \end{pmatrix}$$

$$r^\pi = \begin{pmatrix} r(0,\pi(0)) \\ \vdots \\ r(n-1,\pi(n-1)) \end{pmatrix}$$

$$v^\pi = r^\pi + \gamma P^\pi v^\pi$$

$$\downarrow$$

$$(I - \gamma P^\pi) v^\pi = r^\pi$$

CORE

Control + Optimization Research Lab

Solving Tabular MDPs – Policy Iteration

```python
def PI(P, R, gamma):
    """
    implementation of policy iteration
    """
    nS, nA = R.shape

    # initialize policy
    pi = np.random.randint(nA, size=nS)

    while True:
        v = eval_policy(nS, P, R, gamma, pi)
        pi_next = greedy(P, R, gamma, v)
        if (pi_next == pi).all():
            break
        pi = pi_next

    return v, pi
```

terminal condition : $\pi_{k+1} = \pi_k$

Solving Tabular MDPs – Linear Programming

We use **scipy linear programming** library to solve our problem:

```
from scipy.optimize import linprog
```

$$\min 1^\top v$$

$$s.t.$$

$$\gamma \sum_{s'} p(s'|s,a)v(s') - v(s) \le -r(s,a) \quad \text{for all } s, a.$$

```
def LP(P, R, gamma):

    nS, nA = R.shape
    Id = np.tile(np.identity(nS), reps=(nA, 1))
    A = gamma * P - Id
    b = -np.reshape(R, newshape=(nS * nA, 1), order='F')
    c = 1.0 * np.ones(nS)

    res = linprog(c, A, b, bounds=(None, None))

    v = np.reshape(res['x'], newshape=(nS, 1))
    pi = pi = greedy(P, R, gamma, v)

    return v, pi
```

Fortran-like index order