Discussion 12: Model Free Reinforcement Learning

Terms

Policy

Value Function (State-value, Action-Value)

States

Action

Reward

Episodes

Solving RL problems

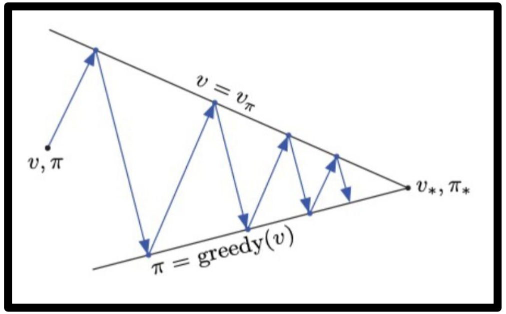
Solving RL problems with the techniques seen so far is an iterative process with two steps

Policy Evaluation

Compute v(s),q(s,a)

Policy Improvement

Compute π(a|s)



Last week: Dynamic Programming approach

Last week we when we computed the value function we assumed we used the following expression.

$$\boxed{v_{k+1}(s)} = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\boxed{\mathcal{R}_s^a} + \gamma \sum_{s' \in \mathcal{S}} \boxed{\mathcal{P}_{ss'}^a} \boxed{v_k(s')} \right)$$

Dynamic Programming

```python
def compute_state_value(max_iter=9, discount=1.0, policy = actions_prob*np.ones((grid_size,grid_size,4))):
    new_state_values = np.zeros((grid_size, grid_size))
    iteration = 0


    while iteration<=max_iter:
      state_values = new_state_values.copy()
      old_state_values = state_values.copy()

      for i in range(grid_size):
          for j in range(grid_size):
              value = 0
              for k,a in enumerate(actions):
                  (next_i, next_j), reward = step([i, j], a)
                  value += policy[i,j,k] * (reward + discount * state_values[next_i, next_j])
              new_state_values[i, j] = value

      iteration += 1

    return new_state_values, iteration
```

This week we consider that we cannot iterate over every action to see the new states and rewards as we did last week

Today: *Episodic Methods*
Monte Carlo (MC) and Temporal Difference (TD) methods

Solving RL problems

Solving RL problems with the techniques seen so far is an iterative process with two steps

Policy Evaluation

Compute $v(s), q(s,a)$

$\rightleftarrows$

Policy Improvement

Compute $\pi(a|s)$

Monte Carlo and Temporal Difference methods are concerned with the policy evaluation

Action Value Function - Two Key Equations

Definition:

$$\boxed{\begin{gathered} Q_{\pi}(s,a) = \mathbb{E}[G_t | s_t = s, a_t = a] \\ G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \end{gathered}}$$

Bellman Equation:

$$\boxed{Q_{\pi}(s,a) = \gamma * Q_{\pi}(s',a') + R(s,a)}$$

Monte Carlo Methods

Collect states, actions, rewards from an episode,

Every Timestep (working backwards):

$$G_N = 0$$
$$G_t = R_t + \gamma^* G_{t+1}$$

Every Time (or the first time) you visit the State Action Pair S,A:

$$N(S_t, A_t) \leftarrow N(S_t, A_t) + 1$$
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{1}{N(S_t, A_t)}(G_t - Q(S_t, A_t))$$

or

$$T(S_t,A_t) \leftarrow T(S_t,A_t) + G_t$$
$$Q(S_t,A_t) \leftarrow T(S_t,A_t)/N(S_t,A_t)$$

Note: These two are mathematically equivalent but the former is cleaner and more memory efficient while the later is more intuitive

SARSA (TD method)

Collect states, actions, rewards from an episode,

Every Time you Visit the State Action Pair S,A:

$$\boxed{Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]}$$

“Model Free” MDP framework

We assume that the environment can be described by

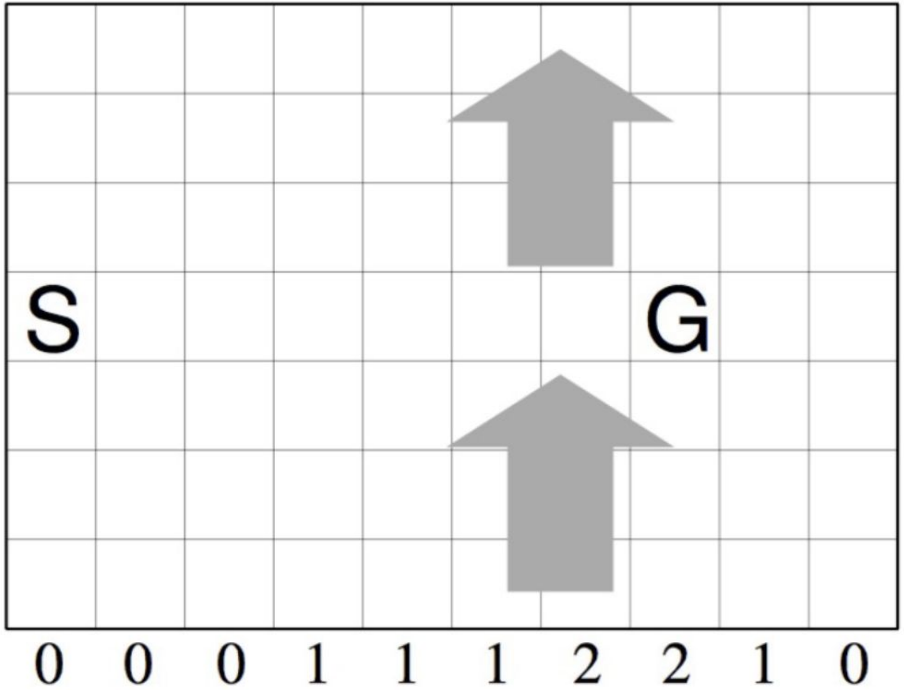
- State set $\mathcal{S}$
- Action set $\mathcal{A}$
- Reward set $\mathcal{R}$
- With dynamics given by a set of probabilities $\mathcal{P}$

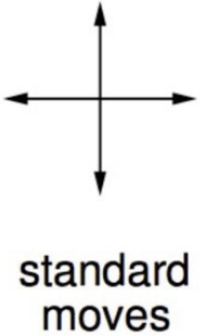
You no longer need to know these explicitly, You just have to be able to generate samples from them

Windy gridworld example

Reward = −1

Undiscounted





Code this week. Implement Sarsa and TD methods

Code overview in colab

Your task is to implement the TD Method

Explore the influence gamma and epsilon have on the policy

Explore how many episodes you need to run to get the optimal policy

Analyze the policy and value function, are there any inconsistencies?