Q-value methods for reinforcement learning

1) SARSA
2) Q-learning

Value functions

- To learn an optimal policy (learn how to act), we need *value functions*
- Two types of value functions in RL:
  - **state value function**, denoted V(s)
  - **state-action value function**, denoted Q(s,a) ← Our focus today

Bellman equation:

$$Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^{\pi}(s',a') \right]$$

Q-value methods

Learning algorithm:

- Initialize $Q(s,a) = 0$ for all (s,a) pairs
- Repeat **episodes (e.g. “games")**
  - select s, a
  - Repeat **steps within episode**
    - execute action a, observe r, s'
    - select a' based on $Q(s', a^*)$ **ϵ-greedy over a^***
    - update $Q(s,a)$
    - $s = s'$, $a = a'$
  - Until s terminal (where $Q(s', a') = 0$)

Update rule differs for SARSA and Q-learning

Q-value update rules

SARSA update rule:

$$\Delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)]$$

On-policy

Q-learning update rule:

$$\Delta Q_t(s_t, a_t) = \alpha[r_{t+1} + \gamma max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)]$$

Off-policy

Tic-tac-toe (in Python)

	0	1	2
0	O	-	-
1	-	X	-
2	-	-	-

- $9^3 = 729$ states
  - 9 board positions
  - Each position has 1 of 3 values: {-, 0, X}
  - States are represented by a string: “0----X----”
- 9 actions (at most)
  - Actions are represented by a tuple: (row, col)
- Q values
  - Q is a dictionary of dictionaries
  - Q[a][s] <-- index value for action **a**, state **s**
  - Initialize all values to 0