UMassAmherst
The Commonwealth's Flagship Campus

Reinforcement Learning- Q-learning and SARSA

Chaitra Gopalappa

References

- Sutton and Barto textbook
  - Chapter 6: Sections 6.4 and 6.5

Overview – RL algorithms in this slide set

- Temporal difference methods
  - Q-learning: off-policy (update independent of policy being followed)
  - SARSA: on-policy (update dependent on policy being followed)

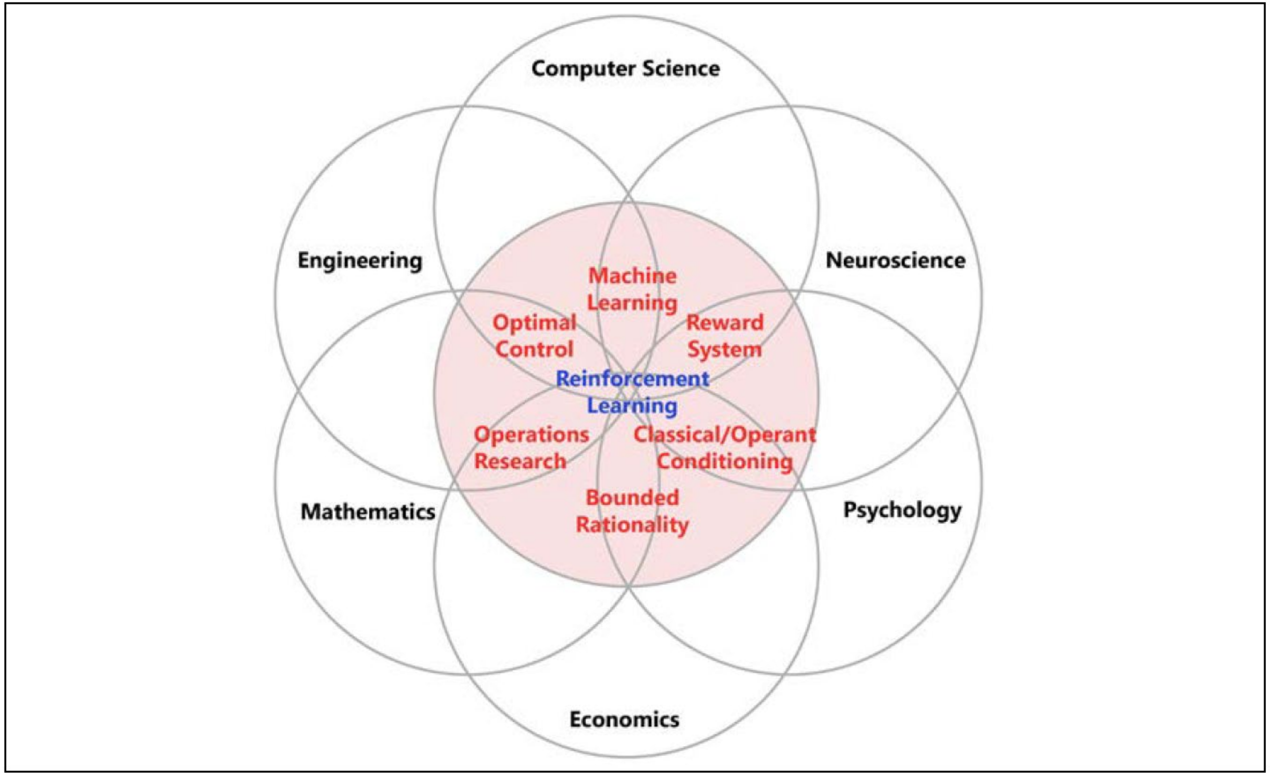


Figure 1.3: Various domains in RL

- Source: DRL- Maxim Lapan, 2nd editon

Recollect:
Theory behind RL: Transitioning from DP to RL

1. Bellman optimality equation to derive Q-factors
2. Q-factor version of Bellman optimality equation
3. Q-factor value iteration – still DP domain- but updating Q-factors instead of value function
4. Robbins- Monro algorithm – to estimate mean of RV from sample
5. Using Robbins-Monro to update Q-factors- leads to 'model-free' algorithm
6. Finally, leads to Q-learning value iteration algorithm

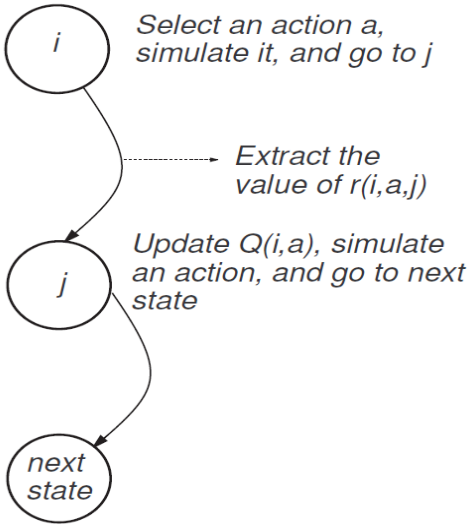
Q-learning overview

Recollect Q-values (action-value functions)

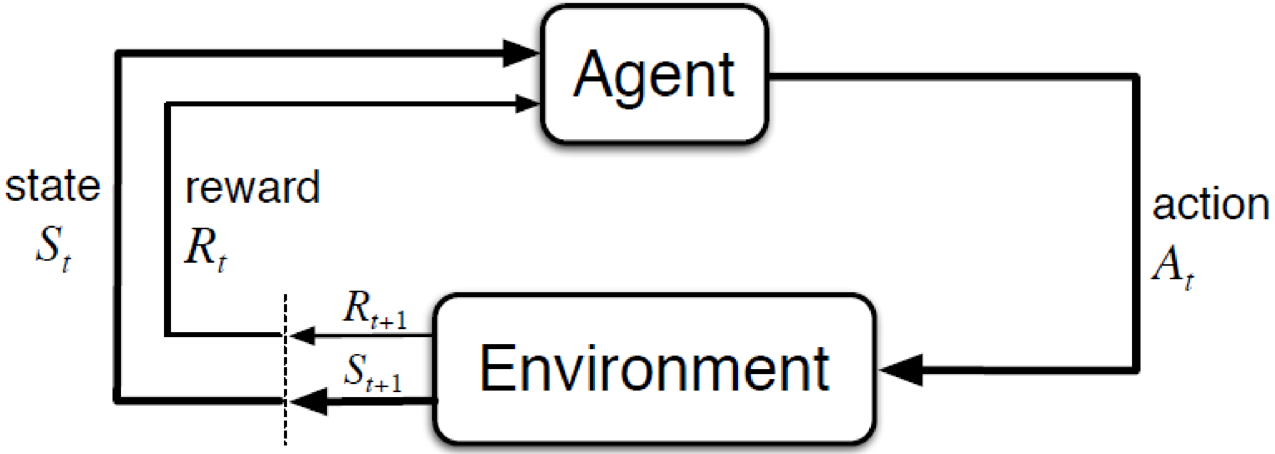
$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\left[R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$$

Start at some state,

1. select action using action selection method,
2. simulate state transition and observe r,
3. update Q-value,
4. check for convergence; if not converged, goto step 1



General reinforcement learning overview



Q-learning – tabular method
(also called off-policy temporal difference (TD) control)

- Set step size: $\alpha \in (0,1], small\ \epsilon > 0$
- Set episode length; If there is absorbing state, episode ends if reaches terminal state or reaches episode length
- Initialize $Q(s,a), \forall(s,a) pairs$; except $Q(terminal\ state, .) = 0$
- Loop for each episode
  - Initialize state, say s
  - Loop for each step of episode (until episode length or terminal state)
    - Choose action a given state s, using **action selection method**
    - Take action a and observe r, s'
    - Update $Q(s,a) = Q(s,a) + \alpha\left[r(s,a,s') + \lambda \max_{a'} Q(s',a') - Q(s,a)\right]$
    - Set $s \leftarrow s'$
    - **Update α**

Action selection method:
Select each action with probability $p_k = \frac{1}{|A|}$

Update α

$\alpha = \frac{1}{k+1}; \alpha = \frac{A}{B+k}; \alpha = \frac{\log k}{k}$

Notice: as k increases weight given to new sample decreases

Q-learning
(also called off-policy temporal difference (TD) control)

- Set step size: $\alpha \in (0,1], small\ \epsilon > 0$
- Set episode length; If there is absorbing state, episode ends if reaches terminal state or reaches episode length
- Initialize $Q(s,a), \forall (s,a) pairs$; except $Q(terminal\ state, .) = 0$
- Loop for each episode
  - Initialize state, say s
  - Loop for each step of episode (until episode length or terminal state)
    - Choose action a given state s, using **action selection method**
    - Take action a and observe r, s'
    - Update $Q(s,a) = Q(s,a) + \alpha\left[r(s,a,s') + \lambda \max_{a'} Q(s',a') - Q(s,a)\right]$
    - Set $s \leftarrow s'$
    - **Update α**

Action selection method:
Then equal selection $p_k = \frac{1}{|A|}$ will not work

Update α
$\alpha = constant$

Action selection methods (Chapter 21, Lapan)

1. Equal chance of selecting action in iteration k: $p_k = \frac{1}{|A|}$ (as per Robbins-Monro, if each pair (i, a) is tried a large number of times, solution would converge, when using non-constant step-size)
2. Greedy-action selection
   1. Let $a^* = argmax_{a \in A}\{Q(s,a)\}$
   2. Then, exploit , i.e., select a^* with probability $(1 - \epsilon)$, and explore with probability ϵ, i.e., equal chance of selecting from all actions
   3. Value of ϵ
      1. Constant small ϵ,
      2. Or vary with iteration number, e.g.,

   $\epsilon = \frac{0.5}{k}$ or $\epsilon = \frac{\frac{b}{V_k(s)}}{k}$; $0 < b < 1$; $V_k(s)$=Number of times state s has been visited
3. Action selection methods: Focus on how much and how to explore v exploit
   1. **Chapter 21, Lapan** discusses other exploration methods, outside of greedy action selection

Move robot on chutes and ladder: Objective is to reach 100 in shortest steps

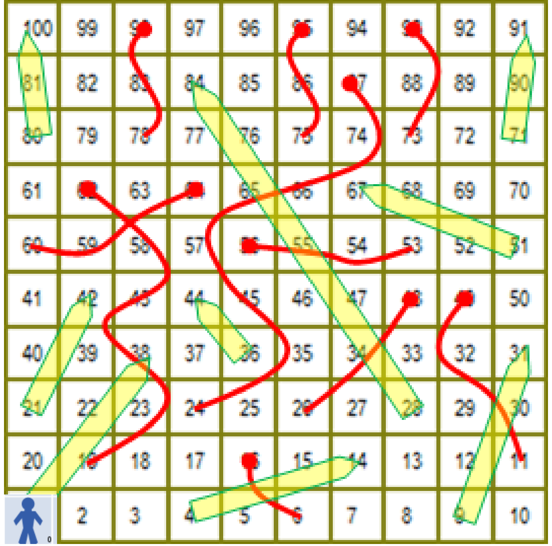
Initialize Q to arbitrary values

Loop for a large number of episodes

1. *Set* $s = 1$
2. *Loop till end of episode*
   1. *Select action, e.g., apply epsilon-greedy*
   2. *Apply action, and determine next state* s'
   3. *Update*

$$Q(s,a) = Q(s,a) + \alpha \left[r(s,a,s') + \lambda \max_{a'} Q(s',a') - Q(s,a) \right]$$

5. *Go to step 1*



$$S = \{1, \dots, 100\}$$

$$A = \{Left, Right, Up, Down\}$$

$$r(s,a,s') = \begin{cases} 1000 \; is \; s' = 100 \\ 0 \; otherwise \end{cases}$$

$\boldsymbol{Q}$: matrix of size 100×4

SARSA $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$
(also called on-policy temporal difference (TD) control)

- Set step size: $\alpha \in (0,1], small\ \epsilon > 0$
- Set episode length; If there is absorbing state, episode ends if reaches terminal state or reaches episode length
- Initialize $Q(s,a), \forall(s,a) pairs$; except $Q(terminal\ state, .) = 0$
- Loop for each episode (until end of episode)
  - Initialize state, say s
  - Choose action a given state s, using **action selection method**
  - Loop for each step of episode
    - Take action a and observe r, s'
    - Choose action a' given state s', using **action selection method**
    - Update $Q(s,a) = Q(s,a) + \alpha[r(s,a,s') + \lambda Q(s',a') - Q(s,a)]$
    - Set $s \leftarrow s', a \leftarrow a'$

Temporal difference methods

- Q-learning: off-policy (update independent of policy being followed)
  - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha \left[R_{t+1} + \lambda \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right]$
  - $Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha \left[R_{t+1} + \lambda \max_a Q(S_{t+1}, a)\right]$
  - Faster convergence
  - Requires ϵ to be gradually reduced otherwise it may not converge
- SARSA: on-policy (update dependent on policy being followed)
  - $Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \lambda\, Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)]$
  - SARSA converges with probability 1 to an optimal policy and action-value function, as long as all state-action pairs are visited an infinite number of times and the policy converges in the limit to the greedy policy.
  - This can be done by selection of appropriate hyperparamters, e.g., $\epsilon = \frac{1}{k}$ in epsilon-greedy action selection method
- Both equivalent of value iteration in DP, i.e., finds an optimal policy
  - There are policy iteration equivalent TD methods too (evaluating value for a fixed policy, and then doing policy improvement)
  - Sutton and Barto Chapter 6

- Robbin-Monro:
  - $X_{k+1} = X_k + \frac{1}{k+1}[x_{k+1} - X_k]$
  - $X_{k+1} = X_k + \alpha[x_{k+1} - X_k] = (1-\alpha)X_k + \alpha \underbrace{x_{k+1}}_{\text{sample}}$

- Q-learning
  - $Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha\left[\overbrace{R_{t+1} + \lambda \max_a Q(S_{t+1}, a)}^{\text{sample}}\right]$

Sample of current (s, a)

Sample of the trajectory for (s, a)

- Robbin-Monro:
  - $X_{k+1} = X_k + \frac{1}{k+1}[x_{k+1} - X_k]$
  - $X_{k+1} = X_k + \alpha[x_{k+1} - X_k] = (1-\alpha)X_k + \alpha \underbrace{x_{k+1}}_{\text{sample}}$

- Q-learning
  - $Q(S_t, A_t) = (1-\alpha)Q(S_t, A_t) + \alpha \Big[\overbrace{R_{t+1} + \lambda \max_a Q(S_{t+1}, a)}^{\text{sample}}\Big]$

Sample of current (s, a)

Sample of the trajectory for (s, a)

UMassAmherst
The Commonwealth's Flagship Campus