Passive Reinforcement Learning

- How to learn from already given experiences
- Still assume an MDP

$$\begin{aligned}
&\text{set of states } s \in S \\
&\text{set of actions (per state) } A \\
&\text{model } T(s, a, s') \to \text{probability} \\
&\text{reward function } R(s, a, s') \\
&\text{looking for a policy } \pi(s)
\end{aligned}$$

- Don't know T or R - Can be thought of as 'online learning' - Passive $\implies$ input = fixed policy $\pi(s)$ - Given a stream of transitions: $\{(s, \pi(s), s', R), (s', \pi(s'), s'', R'), ...\}$ - Goal: learn the state values $V(s)$

Model-Based Learning

- Idea: learn an approximate model based on experiences
- Solve for values as if the learned model were correct
- Steps:
  1. Learn empirical MDP model
     - Count outcome's s' for each (s, a)
     - Normalize all $T(s, a, s')$ to give an estimate of $\hat{T}(s, a, s')$
     - Discover each $\hat{R}(s, a, s')$ when we experience (s, a, s')
  2. Solve the learned MDP
     - For example, use value iteration, as before
- If you know $\mathbb{P}(A)$, $\mathbb{E}[A] = \sum_a a\mathbb{P}(a)$
- If you don't know $\mathbb{P}(A)$,

$$\hat{\mathbb{P}}(a) = \frac{|a|}{N}$$
$$\mathbb{E}[A] \approx a\hat{\mathbb{P}}(a)$$

  - Have to collect N samples $\{a_1, a_2 ..., a_N\}$

Model Free

- If you don't know $P(A)$, $\mathbb{E}[A] \approx \frac{1}{N}\sum_i a_i$
- Goal: compute values for each state under π
- Direct / Monte Carlo evaluation
  - Every time you visit a state, write down what the sum of discounted rewards turned out to be from that state until the end of the episode / game

$$\text{sample}_\text{i}(s) = R(s) + \gamma R(s') + \gamma^2 R(s'') + ...$$
$$V(s) = \frac{1}{N}\sum_i \text{sample}_\text{i}(s)$$

  - Easy to understand
  - Doesn't require any knowledge of T or R
  - Eventually computes the correct average values
  - Doesn't preserve state connections
- Sample-Based Policy Evaluation
  - Take samples of outcomes' s' (by doing the action) and average

$$\begin{aligned}
&\text{sample}_1 = R(s, \pi(s), s'_1 + \gamma V^\pi_k(s'_1)) \\
&\text{sample}_2 = R(s, \pi(s), s'_2 + \gamma V^\pi_k(s'_2)) \\
&\vdots \\
&\text{sample}_n = R(s, \pi(s), s'_n + \gamma V^\pi_k(s'_n)) \\
\\
&V^\pi_{k+1}(s) = \frac{1}{n}\sum_i \text{sample}_i
\end{aligned}$$

- Temporal difference learning of values
  - Policy still fixed, still doing evaluation
  - move values toward value of whatever successor occurs: running average
  - $$\begin{gathered}\text{Sample of } V(s) \hat{=} \text{sample} = R(s, \pi(s), s') + \gamma V^{\pi}(s') \\ \text{Update to } V(s) \hat{=} V^{\pi}(s) = (1+\alpha)V^{\pi}(s) + \alpha \times \text{sample}\end{gathered}$$
  - Exponential Moving Average (for the update)
    - The running interpolation update: $\bar{x}_n = (1-\alpha)\bar{x}_{n-1} + \alpha x_n \mid_{0<\alpha<1}$
    - Makes recent samples more important
    - Decreasing learning rate α can give converging averages
  - Good to do policy evaluation, but hard to turn into a new policy
    - $$\begin{aligned}&\pi(s) = \arg\max_a Q(s,a) \\ &Q(s,a) = \textstyle\sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V(s')]\end{aligned}$$
    - Use Q values:

$$Q_k(s,a) = \begin{cases} 0, k = 0 \\ Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')] \end{cases}$$

    - sample $= R(s,a,s') + \gamma \max_{a'} Q(s',a')$
    - $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[\text{sample}]$
    - Converge to the optimal policy even if you're acting sub-optimally (called off-policy learning)