Reinforcement Learning

- Reinforcement learning:
  - Still assume an MDP:
    - A set of states \( s \in S \)
    - A set of actions (per state) \( A \)
    - A model \( T(s,a,s') \)
    - A reward function \( R(s,a,s') \)
  - Still looking for a policy \( \pi(s) \)

  - New twist: don’t know \( T \) or \( R \)
    - I.e. don’t know which states are good or what the actions do
    - Must actually try actions and states out to learn

Passive Learning

- Simplified task
  - You don’t know the transitions \( T(s,a,s') \)
  - You don’t know the rewards \( R(s,a,s') \)
  - You are given a policy \( \pi(s) \)
  - Goal: learn the state values
  - ... what policy evaluation did

  In this case:
  - Learner “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…

Example: Direct Evaluation

- Episodes:
  - \( (1,1) \) up -1
  - \( (1,2) \) up -1
  - \( (1,3) \) right -1
  - \( (2,3) \) right -1
  - \( (3,3) \) right -1
  - \( (3,2) \) up -1
  - \( (3,1) \) right -1
  - \( (4,3) \) exit +100

\[ \gamma = 1, R = -1 \]
\[ V(2,3) = (96 + -103) / 2 = -3.5 \]
\[ V(3,3) = (99 + 97 + -102) / 3 = 31.3 \]

Recap: Model-Based Policy Evaluation

- Simplified Bellman updates to calculate \( V \) for a fixed policy:
  - New \( V \) is expected one-step look-ahead using current \( V \)
  - Unfortunately, need \( T \) and \( R \)

\[ V_0^\pi(s) = 0 \]
\[ V_{i+1}^\pi(s) \leftarrow \sum_{a'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i^\pi(s')] \]

Model-Based Learning

- Idea:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct

- Simple empirical model learning
  - Count outcomes for each \( s,a \)
  - Normalize to give estimate of \( T(s,a,s') \)
  - Discover \( R(s,a,s') \) when we experience \( (s,a,s') \)

- Solving the MDP with the learned model
  - Iterative policy evaluation, for example

\[ V_{i+1}^\pi(s) \leftarrow \sum_{a'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V_i^\pi(s')] \]
### Sample-Based Policy Evaluation?

$V_{n+1}^\pi(s) = \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V_n^\pi(s') \right]$

- Who needs $T$ and $R$? Approximate the expectation with samples (drawn from $T$)
  
  - $sample_1 = R(s, \pi(s), s_1') + \gamma V_n^\pi(s_1')$
  - $sample_2 = R(s, \pi(s), s_2') + \gamma V_n^\pi(s_2')$
  - $\ldots$
  - $sample_k = R(s, \pi(s), s_k') + \gamma V_n^\pi(s_k')$

$V_{n+1}^\pi(s) = \frac{1}{k} \sum_{i=1}^{k} sample_i$

Almost! But we only actually make progress when we move to $n+1$.

### Temporal-Difference Learning

- Big idea: learn from every experience!
  - Update $V(s)$ each time we experience $(s, a, s', r)$
  - Likely $s'$ will contribute updates more often

- Temporal difference learning
  - Policy still fixed
  - Move value toward value of whatever successor occurs: running average!

Sample of $V(s)$: $sample = R(s, \pi(s), s') + \gamma V(s')$

Update to $V(s)$:

$V(s) \leftarrow V(s) - (1 - \alpha) V(s) + \alpha \cdot sample$

Same update:

$V(s) \leftarrow V(s) - V(s) + \alpha \cdot sample$

### Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important
    
    $E_n = \sum_{m=0}^{\infty} (1 - \alpha)^m \cdot s_{m-1} + (1 - \alpha)^2 \cdot s_{m-2} + \ldots$
    
    - $1 + (1 - \alpha) + (1 - \alpha)^2 + \ldots$

- Forgets about the past (distant past values were wrong anyway)
- Easy to compute from the running average
  
  $E_n = (1 - \alpha)^n \cdot E_0 + \alpha \cdot s_n$

- Decreasing learning rate can give converging averages
Problems with TD Value Learning

- TD value learning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we're sunk:
  \[
  \pi(s) = \arg \max_a Q^*(s,a)
  \]
  \[
  Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]
  \]
- Idea: learn Q-values directly
- Makes action selection model-free too!

Active Learning

- Full reinforcement learning
  - You don't know the transitions \( T(s,a,s') \)
  - You don't know the rewards \( R(s,a,s') \)
  - You can choose any actions you like
  - Goal: learn the optimal policy
  - … but value iteration did!
- In this case:
  - Learner makes choices!
  - Fundamental tradeoff: exploration vs. exploitation
  - This is NOT offline planning! You actually take actions in the world and find out what happens...

Detour: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with \( V_0(s) = 0 \), which we know is right (why?)
  - Given \( V_i \), calculate the values for all states for depth \( i+1 \):
    \[
    V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V_i(s') \right]
    \]
- But Q-values are more useful!
  - Start with \( Q_0(s,a) = 0 \), which we know is right (why?)
  - Given \( Q_i \), calculate the q-values for all q-states for depth \( i+1 \):
    \[
    Q_{i+1}(s,a) \leftarrow \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q_i(s',a') \right]
    \]

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but don’t decrease it too quickly!
  - Basically doesn’t matter how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)

Q-Learning

- Q-Learning: sample-based Q-value iteration
- Learn \( Q^*(s,a) \) values
  - Receive a sample \( (s,a,s',r) \)
  - Consider your old estimate: \( Q(s,a) \)
  - Consider your new sample estimate:
    \[
    Q'(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]
    \]
  - Incorporate the new estimate into a running average:
    \[
    Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha [\text{sample}]
    \]

Exploration / Exploitation

- Several schemes for forcing exploration
  - Simplest: random actions (\( \varepsilon \)-greedy)
    - Every time step, flip a coin
      - With probability \( \varepsilon \), act randomly
      - With probability 1-\( \varepsilon \), act according to current policy
  - Problems with random actions?
    - You do explore the space, but keep thrashing around once learning is done
    - One solution: lower \( \varepsilon \) over time
    - Another solution: exploration functions
Exploration Functions

- **When to explore**
  - Random actions: explore a fixed amount
  - Better idea: explore areas whose badness is not (yet) established

- **Exploration function**
  - Takes a value estimate and a count, and returns an optimistic utility, e.g. $f(u, n) = u + k/n$ (exact form not important)

$$Q_{t+1}(s, a) = R(s, a, s') + \gamma \max_{a'} Q_t(s', a')$$

$$Q_{t+1}(s, a) = R(s, a, s') + \gamma \max_{a'} f(Q_t(s', a'), N(s', a'))$$

Q-Learning

- Q-learning produces tables of q-values: