Recap: Defining MDPs

- **Markov decision processes:**
  - Set of states $S$
  - Start state $s_0$
  - Set of actions $A$
  - Transitions $P(s' | s, a)$ (or $T(s, a, s')$)
  - Rewards $R(s, a, s')$ (and discount $\gamma$)

- **MDP quantities so far:**
  - Policy = Choice of action for each state
  - Utility = sum of (discounted) rewards
Solving MDPs

- Value Iteration
  - Asynchronous VI
- Policy Iteration
- Reinforcement Learning

\[ V^* = \text{Optimal Value Function} \]

The value (utility) of a state \( s \):

\[ V^*(s) \]

“expected utility starting in \( s \) & acting optimally forever”
<table>
<thead>
<tr>
<th>Q*</th>
<th>The value (utility) of the q-state (s,a):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( Q^*(s,a) )</td>
</tr>
<tr>
<td></td>
<td>“expected utility of 1) starting in state ( s )</td>
</tr>
<tr>
<td></td>
<td>2) taking action ( a )</td>
</tr>
<tr>
<td></td>
<td>3) acting optimally forever after that”</td>
</tr>
<tr>
<td></td>
<td>( Q^*(s,a) = \text{reward from executing } a \text{ in } s \text{ then ending in } s' )</td>
</tr>
<tr>
<td></td>
<td>plus... discounted value of ( V^*(s') )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \pi^* )</th>
<th>Specifies The Optimal Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi^*(s) = \text{optimal action from state } s )</td>
<td></td>
</tr>
</tbody>
</table>
The Bellman Equations

How to be optimal:
Step 1: Take correct first action
Step 2: Keep being optimal

Definition of “optimal utility” via expectimax recurrence gives a simple one-step lookahead relationship amongst optimal utility values

\[ V^*(s) = \max_a Q^*(s, a) \]

\[ Q^*(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V^*(s') \right] \]

These are the Bellman equations, and they characterize optimal values in a way we’ll use over and over.
Gridworld: \( Q^* \)

Q-VALUES AFTER 100 ITERATIONS

Gridworld Values V* \( V^*(s) = \max_a Q^*(s, a) \)

VALUES AFTER 100 ITERATIONS
No End in Sight...

- We’re doing way too much work with expectimax!

- Problem 1: States are repeated
  - Idea: Only compute needed quantities once
  - Like graph search (vs. tree search)

- Problem 2: Tree goes on forever
  - Rewards @ each step → V changes
  - Idea: Do a depth-limited computation, but with increasing depths until change is small
  - Note: deep parts of the tree eventually don’t matter if $\gamma < 1$

Time-Limited Values

- Key idea: *time-limited values*

- Define $V_\ell(s)$ to be the optimal value of $s$ if the game ends in $\ell$ more time steps
  - Equivalently, it’s what a depth-$\ell$ expectimax would give from $s$
Value Iteration

- For all states, initialize $V_0(s) = 0$ (no time steps left means an expected reward of zero)
- Repeat
  - $K += 1$
  - $Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') [ R(s, a, s') + \gamma V_k(s') ]$
  - $V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

- Repeat until $|V_{k+1}(s) - V_k(s)| < \epsilon$, for all states “convergence”

Called a “Bellman Backup”

Successive approximation; dynamic programming
Example: Value Iteration

Assume no discount ($\gamma=1$) to keep math simple!

$$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$$

$$V_{k+1}(s) = \max_a Q_{k+1}(s, a)$$

Example: Value Iteration

Assume no discount ($\gamma=1$) to keep math simple!

$$V_0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$V_1 = \begin{bmatrix} \text{Value} \end{bmatrix}$$

$$V_2 = \begin{bmatrix} \text{Value} \end{bmatrix}$$

$$Q_{k+1}(s, a) = \Sigma_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$$

$$V_{k+1}(s) = \max_a Q_{k+1}(s, a)$$
Example: Value Iteration

$Q(s, a)$

$V_0$

$V_1$

$V_2$

$Q_k(s, a) = \Sigma_s T(s, a, s') [ R(s, a, s') + \gamma V_k(s') ]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$

Assume no discount ($\gamma = 1$) to keep math simple!
### Example: Value Iteration

<table>
<thead>
<tr>
<th>$V_0$</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_1(s,a)$</td>
<td>1, 2</td>
<td>1, -10</td>
<td>0</td>
</tr>
<tr>
<td>$V_1$</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$Q_2(s,a)$</td>
<td>3, 3.5</td>
<td>2.5, -10</td>
<td>0</td>
</tr>
<tr>
<td>$V_2$</td>
<td>3.5</td>
<td>2.5</td>
<td>0</td>
</tr>
</tbody>
</table>

$Q_k+1(s, a) = \Sigma_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]$

$V_{k+1}(s) = \text{Max}_a Q_{k+1}(s, a)$
If agent is in 4,3, it only has one legal action: get jewel. It gets a reward and the game is over. If agent is in the pit, it has only one legal action, die. It gets a penalty and the game is over. Agent does NOT get a reward for moving INTO 4,3.
$k=2$

VALUES AFTER 2 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

$k=3$

VALUES AFTER 3 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
$k=6$

VALUES AFTER 6 ITERATIONS

VALUES AFTER 7 ITERATIONS

$k=7$
**k=8**

VALUES AFTER 8 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

**k=9**

VALUES AFTER 9 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0
k=10

VALUES AFTER 10 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0

k=11

VALUES AFTER 11 ITERATIONS

Noise = 0.2
Discount = 0.9
Living reward = 0
**k=12**

VALUES AFTER 12 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0

**k=100**

VALUES AFTER 100 ITERATIONS

Noise = 0.2  
Discount = 0.9  
Living reward = 0
Let’s imagine we have the optimal values $V^*(s)$

How should we act?
- In general, it’s not obvious!

We need to do a mini-expectimax (one step)

$$
\pi^*(s) = \arg \max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')]
$$

This is called policy extraction, since it gets the policy implied by the values.
Computing Actions from Q-Values

- Let’s imagine we have the optimal q-values:

- How should we act?
  - Completely trivial to decide!

  \[ \pi^*(s) = \arg \max_a Q^*(s, a) \]

- Important lesson: actions are easier to select from q-values than values!

---

Value Iteration - Recap

- **For all** \( s \), Initialize \( V_0(s) = 0 \)  
  *no time steps left means an expected reward of zero*

- **Repeat**
  - \( K := 1 \)
  - **Repeat** for all states, \( s \), and all actions, \( a \):

    \[
    Q_{k+1}(s, a) = \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_k(s') \right]
    \]

    \[
    V_{k+1}(s) = \max_a Q_{k+1}(s, a)
    \]

- **Until** \( |V_{k+1}(s) - V_k(s)| < \epsilon \), **for all** \( s \)  
  *“convergence”*

- **Theorem:** will converge to unique optimal values