Outline

- Last class:
  - Markov chains
  - Markov Decision Processes
  - Policies and value functions
    - Policy evaluation
    - Policy improvement
  - Policy iteration algorithm
- Today:
  - Value iteration
  - Function approximation
  - Reinforcement learning
Markov Decision Processes (MDPs)

- Set of states \( S \)
- Set of actions \( A \)
- Reward function \( R: S \times A \rightarrow \mathbb{R} \)
  - \( R(s,a) \) is the short-term utility of the action.
- Transition model (dynamics): \( T: S \times A \times S \rightarrow [0,1] \)
  - \( T(s,a,s') = P(s_{t+1} = s' | s_t = s, a_t = a) \) is the probability of going from \( s \) to \( s' \) under action \( a \).
- Discount factor, \( \gamma \) (between 0 and 1, usually close to 1).

Policy Iteration Algorithm

- Start with an initial policy \( \pi_0 \) (e.g. random)
- Iterate:
  - Compute \( V^\pi \), using policy evaluation:
    - \( V_{k+1}(s) \leftarrow (R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_k(s')) \)
  - Compute a new policy \( \pi' \) that is greedy with respect to \( V^\pi \):
    - Do once: \( \pi'(s) = \arg\max_{a \in A} (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s')) \)
- Terminate when \( V^\pi = V^{\pi'} \)
A 4x3 gridworld example

- Problem description:
  - 11 discrete states, 4 motion actions (N, S, E, W) in each state.
  - Transitions are deterministic. If direction is blocked, agent stays in same location.
  - Reward is +1 in top right state, -10 in state directly below, -0 elsewhere.
  - Episode terminates when the agent reaches +1 or -10 state.
  - Discount factor $\gamma = 0.99$.

- What is $V^\pi(s)$ for $\pi(s) = \text{Up}, \forall s$?
  - $V^\pi(s)=0$ for most states, except right column.

A 4x3 gridworld example

- New version:
  - Transitions are now stochastic, as shown on left figure.
Policy Iteration (1)

Policy Iteration (2)
Policy Iteration (3)

A 4x3 gridworld example

- New version:
  - Transitions are stochastic.
  - Change the reward of the pit from -10 to -500.

Agent actively tries to avoid the goal, for fear of falling into the pit!
Generalized Policy Iteration

- Any combination of policy evaluation and policy improvement steps, e.g., only update the value of one state, and immediately improve the policy at that state.

Optimal policies and optimal value functions

- The optimal value function $V^*$ is defined as the best value that can be achieved at any state:

  $$ V^*(s) = \max_{\pi} V^\pi(s) $$

- In a finite MDP, there exists a unique optimal value function (shown by Bellman, 1957).

- Any policy that achieves the optimal value function is called an optimal policy (denoted $\pi^*$). The optimal policy is not necessarily unique.
Optimal policies in the gridworld example

- Optimal state values give information about the shortest path to the goal.
- One of the deterministic optimal policies is shown below.
- There can be an infinite number of optimal policies (think stochastic policies).

Complexity of policy iteration

Repeat two basic steps: Compute $V^\pi$ + Compute a new policy $\pi'$

1. Compute $V^\pi$, using policy evaluation.
   Per iteration: $O(S^3)$

2. Compute a new policy $\pi'$ that is greedy with respect to $V^\pi$.
   Per iteration: $O(S^2A)$

Repeat for how many iterations?
   At most $|A|^{|S|}$

*Can get very expensive when there are many states!*
Bellman Optimal Equation for $V^*$

- The value of a state under the optimal policy must be equal to the expected return for the best action in the state:
  
  $$V^*(s) = \max_{a \in A} E \left[ R_t + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right]$$
  
  $$= \max_{a \in A} \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

  $V^*$ is the unique solution of this system of non-linear equations.

- If we know $V^*$ (and $R, T, \gamma$), then we can compute $\pi^*$ easily:
  
  $$\pi^*(s) = \arg\max_{a \in A} \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s') \right)$$

One way to compute $V^*$ is through policy iteration.

Idea: Can we compute $V^*$ directly (without computing $\pi$ at every iteration)?

Value Iteration Algorithm

Main idea: Turn the Bellman optimality equation into an iterative update rule (same as done in policy evaluation):

1. Start with an arbitrary initial approximation $V_0$.

2. On each iteration, update the value function estimate:
   
   $$V_i(s) = \max_{a \in A} \left( R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_{i-1}(s') \right)$$

3. Stop when the max value change between iterations is below a threshold.

The algorithm converges (in the limit) to the true $V^*$. 
Value Iteration (1)

\[
\begin{array}{ccc}
0 & 0 & 0 \\
0 & 0 & -10 \\
0 & 0 & 0 \\
\end{array}
\]

Value Iteration (2)

\[
\begin{array}{ccc}
0 & 0 & 0.69 \\
0 & -0.99 & -10 \\
0 & 0 & -0.99 \\
\end{array}
\]

Bellman residual: \(|V_2(s) - V_1(s)| = 0.99|
### Value Iteration (5)

|         | V_5(s) | V_4(s) | |   |
|---------|--------|--------||---|
| 0.48    | 0.70   | 0.76   | | +1 |
| 0.23    | -0.55  | -10    | |   |
| 0       | -0.23  | -1.40  | |   |

Bellman residual: $|V_5(s) - V_4(s)| = 0.23$

### Value Iteration (20)

|         | V_5(s) | V_4(s) | |   |
|---------|--------|--------||---|
| 0.78    | 0.80   | 0.81   | | +1 |
| 0.77    | -0.44  | -10    | |   |
| 0.75    | 0.69   | 0.37   | | -0.92 |

Bellman residual: $|V_5(s) - V_4(s)| = 0.008$
Compare VI and PI

<table>
<thead>
<tr>
<th></th>
<th>Value Iteration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 iterations</td>
<td></td>
</tr>
<tr>
<td>0.78</td>
<td>0.80</td>
<td>0.81</td>
</tr>
<tr>
<td>0.77</td>
<td>-0.44</td>
<td>-10</td>
</tr>
<tr>
<td>0.75</td>
<td>0.69</td>
<td>0.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-0.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Policy Iteration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 iterations</td>
<td></td>
</tr>
<tr>
<td>0.78</td>
<td>0.80</td>
<td>0.81</td>
</tr>
<tr>
<td>0.77</td>
<td>-0.41</td>
<td>-10</td>
</tr>
<tr>
<td>0.75</td>
<td>0.69</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-0.90</td>
</tr>
</tbody>
</table>

Another example: Four Rooms

- Four actions, fail 30% of the time.
- No rewards until the goal is reached, $\gamma = 0.9$.
- Values propagate backwards from the goal.
Complexity analysis

- Policy iteration:
  - Per iteration: $O(S^2 + S^3A)$
  - $N^2$ iteration: At most $|A|^{|S|}$
  - Fewer iterations

- Value iteration:
  - Per iteration: $O(S^2A)$ (down to $O(SA)$ if few successor states)
  - $N^0$ iterations: Polynomial in $1 / (1 - \gamma)$
  - Faster per iteration

A more efficient VI algorithm

- Instead of updating all states on every iteration, focus on important states.
- Here, we can define important as visited often.
  E.g., board positions that occur on every game, rather than just once in 100 games.
- Asynchronous dynamic programming algorithm:
  - Generate trajectories through the MDP.
  - Update states whenever they appear on such a trajectory.
- This focuses the updates on states that are actually possible.
Limitations of MDPs

1. Finding an optimal policy is polynomial in the number of states.
   - Number of states is often astronomical.
   - Dynamic programming can solve problems up to $10^7$ states.

2. Value iteration and policy iteration assume the model (transitions and rewards) is known in advance.

3. State is sometimes not observable (similar to HMM case).
   - Some states may “look the same”.
   - Sensors data may be noisy.

Solving large MDPs

• For large problems or continuous state space:
  - Define function $V_w(s)$ that approximates the true value function, optimize its parameters $w$.

• Popular solution is to use a linear function: $V_w(s) = \sum_m w_m \phi_m(s)$
  - $\{\phi_1(s), \phi_2(s), ..., \phi_m(s)\}$ are basis features describing information about the state (similar to the evaluation function in games).
  - $w_m$ are weights controlling the linear combination of basis features.

• What is the best weight vector, $w$?

• What is a good set of basis features, $\phi$?
Training the weights from data

Mean Squared Error: \( E_w = \sum_i (y_i - \sum_m w_m \phi_m(x_i))^2 \)

Solve in closed form: \( w_m = (\sum_i \phi_m(x_i)\phi_m(x_i))^{-1}(\sum_i \phi_m(x_i)y_i) \)

Gradient solution: \( w_{m(t+1)} = w_{m(t)} - \alpha_t \sum_i V_{w}(E_{i,w}) E_{i,w} \)

Many cases of non-linear function approximation (e.g. neural networks).

Potential issue: Many fn approximation methods are unstable. Some linear approx. have good results (w/conditions).

Choosing the basis features

- A good basis set is:
  - Reasonably small (easy to guarantee)
  - Fits the value function well (hard to guarantee)

- Suggestions:
  - Use prior knowledge of domain structure to hand-design good features.
  - Search over candidate basis sets (e.g. using hill-climbing), keep the best set.
  - Use a flexible scheme to add/remove basis functions (e.g. genetic algorithms).
Limitations of MDPs

1. Finding an optimal policy is polynomial in the number of states.
   - Number of states is often astronomical.
   - Dynamic programming can solve problems up to $10^7$ states.

2. Value iteration and policy iteration assume the model (transitions and rewards) is known in advance.

2. State is sometimes not observable (similar to HMM case).
   - Some states may “look the same”.
   - Sensors data may be noisy.

What you should know

- Definition of MDP framework.
- Basic MDP algorithms (policy evaluation, policy iteration, value iteration) and their properties.
- Differences/similarities between MDPs and other AI approaches (e.g. general search, game playing, STRIPS planning).
- Function approximation (why, what, how)
Final notes

• **Project code** due today. **Test it on the Trottier machines.**

• **Project report** due tomorrow. Read the instructions closely before finalizing. See the evaluation form on course website.

• There is a 20% penalty (assessed separately for code & report) for submitting anytime after the deadline, up to 5 days.

• **Homework 5** posted. Due Apr. 15.

• **Tutorial 5** on Apr. 12. Utilities, MDPs, reinforcement learning.

• **Course evaluations** available on Minerva.