Today’s Outline

- Reinforcement Learning
  - Q-value iteration
  - Q-learning
  - Exploration/exploitation
  - Linear function approximation

Recap: MDPs

- Markov decision processes:
  - States \( S \)
  - Actions \( A \)
  - Transitions \( T(s,a,s') \) aka \( P(s'|s,a) \)
  - Rewards \( R(s,a,s') \) (and discount \( \gamma \))
  - Start state \( s_0 \) (or distribution \( P_0 \))
- Algorithms
  - Value Iteration
  - Q-value iteration
- Quantities:
  - Policy = map from states to actions
  - Utility = sum of discounted future rewards
  - Q-Value = expected utility from a q-state
    - i.e. from a state/action pair

Bellman Equations

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

\[ Q^*(a,s) = \sum_{s' \in S} P(r(s',a,s) + \gamma V^*(s')) \]

Q-Value Iteration

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

Initialize each q-state: \( Q_0(s,a) = 0 \)

Repeat

For all q-states, \( s,a \)

Compute \( Q_{i+1}(s,a) \) from \( Q_i \) by Bellman backup at \( s,a \).

Until \( \max_{s,a} |Q_{i+1}(s,a) - Q_i(s,a)| < \varepsilon \)

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

Reinforcement Learning

- Markov decision processes:
  - States \( S \)
  - Actions \( A \)
  - Transitions \( T(s,a,s') \) aka \( P(s'|s,a) \)
  - Rewards \( R(s,a,s') \) (and discount \( \gamma \))
  - Start state \( s_0 \) (or distribution \( P_0 \))

- Algorithms
  - Q-value iteration \( \rightarrow \) Q-learning

- Approaches for mixing exploration & exploitation
  - \( \varepsilon \)-greedy
  - Exploration functions

Applications

- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

Stanford Autonomous Helicopter

http://heli.stanford.edu/

Two main reinforcement learning approaches

- Model-based approaches:
  - explore environment & learn model, \( T=P(s'|s,a) \) and \( R(s,a) \), (almost) everywhere
  - use model to plan policy, MDP-style
  - approach leads to strongest theoretical results
  - often works well when state-space is manageable

- Model-free approach:
  - don’t learn a model; learn value function or policy directly
  - weaker theoretical results
  - often works better when state space is large

Two main reinforcement learning approaches

- Model-based approaches:
  - Learn \( T + R \)
  - \(|S|^2|A| + |S||A| \) parameters \((40,000)\)

- Model-free approach:
  - Learn \( Q \)
  - \(|S||A| \) parameters \((400)\)
Recap: Sampling Expectations

- Want to compute an expectation weighted by \( P(x) \):
\[
E[f(x)] = \sum_x P(x) f(x)
\]
- Model-based: estimate \( P(x) \) from samples, compute expectation
\[
\hat{P}(x) = \frac{\text{count}(x)}{k} \quad E[f(x)] \approx \sum_x \hat{P}(x) f(x)
\]
- Model-free: estimate expectation directly from samples
\[
x_i \sim P(x) \quad E[f(x)] \approx \frac{1}{k} \sum_i f(x_i)
\]
- Why does this work? Because samples appear with the right frequencies!

Recap: Exp. Moving Average

- Exponential moving average
- Makes recent samples more important
- Forgets about the past (distant past values were wrong anyway)
- Easy to compute from the running average
\[
x_n = (1-\alpha) \cdot x_{n-1} + \alpha \cdot x_n
\]
- Decreasing learning rate can give converging averages

Q-Learning Update

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

Exploration-Exploitation tradeoff

- You have visited part of the state space and found a reward of 100
  - is this the best you can hope for???
- **Exploitation:** should I stick with what I know and find a good policy w.r.t. this knowledge?
  - at risk of missing out on a better reward somewhere
- **Exploration:** should I look for states w/ more reward?
  - at risk of wasting time & getting some negative reward

Exploration / Exploitation

- Several schemes for action selection
  - Simplest: random actions (\( \epsilon \) greedy)
    - Every time step, flip a coin
    - With probability \( \epsilon \), act randomly
    - With probability \( 1-\epsilon \), 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 \( \epsilon \) over time
  - Another solution: exploration functions

Q-Learning: \( \epsilon \) Greedy
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. \( f(u, n) = u + k/n \)
  - Exploration policy \( \pi(s') = \max_a Q(s', a') \) vs. \( \max_a f(Q(s', a'), N(s', a')) \)

Q-Learning Final Solution

- Q-learning produces tables of q-values:

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - ... but not decrease it too quickly!
  - Not too sensitive to how you select actions (!)

- Neat property: off-policy learning
  - Learn optimal policy without following it (some caveats)

Q-Learning – Small Problem

- Doesn’t work
  - In realistic situations, we can’t possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

- Instead, we need to generalize:
  - Learn about a few states from experience
  - Generalize that experience to new, similar states
    (Fundamental idea in machine learning)

Example: Pacman

- Let’s say we discover through experience that this state is bad:
  - In naïve Q learning, we know nothing about related states and their Q values:
  - Or even this third one!

Feature-Based Representations

- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - \( 1 / (\text{dist to dot})^2 \)
    - Is Pacman in a tunnel? (0/1)
    - ... etc.
  - Can also describe a q-state \((s, a)\) with features (e.g. action moves closer to food)
Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a linear combination of a few weights:
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]
- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but actually be very different in value!

Function Approximation

\[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]

- Q-learning with linear q-functions:
  \[
  \text{transition} = (s, a, r, s') \\
  \text{difference} = r + \max_{s''} Q(s', a) - Q(s, a) \\
  Q(s, a) \leftarrow Q(s, a) + \alpha [\text{difference}] f_i(s, a)
  \]
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state’s features
- Formal justification: online least squares

Example: Q-Pacman

\[ Q(s, a) = 4.0 f_{\text{DOT}}(s, a) - 1.0 f_{\text{GST}}(s, a) \]

- \( f_{\text{DOT}}(s, \text{NORTH}) = 0.5 \)
- \( f_{\text{GST}}(s, \text{NORTH}) = 1.0 \)
- \( Q(s, a) = +1 \)
- \( R(s, a, s') = -500 \)
- \( w_{\text{DOT}} \leftarrow 4.0 + \alpha [-501] 0.5 \)
- \( w_{\text{GST}} \leftarrow -1.0 + \alpha [-501] 1.0 \)
- \( Q(s, a) = 3.0 f_{\text{DOT}}(s, a) - 3.0 f_{\text{GST}}(s, a) \)

Linear Regression

\[
\hat{y} = w_0 + w_1 f_1(x) + w_2 f_2(x)
\]

Ordinary Least Squares (OLS)

\[
\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \sum_k w_k f_k(x_i))^2
\]

Minimizing Error

Imagine we had only one point \( x \) with features \( f(x) \):

\[
\hat{w} = \arg \min_{w} \left( \frac{1}{2} \left( y - \sum_k w_k f_k(x) \right)^2 \right)
\]

\[
\frac{\partial}{\partial w_m} \text{error}(w) = -\left( y - \sum_k w_k f_k(x) \right) f_m(x)
\]

\[
w_m \leftarrow w_m + \alpha \left[ y - \sum_k w_k f_k(x) \right] f_m(x)
\]

Approximate q update:

\[
w_m \leftarrow w_m + \alpha \left[ r + \gamma \max_{s'} Q(s', a') - Q(s, a) \right] f_m(s, a)
\]
Overfitting

Which Algorithm?
Q-learning, no features, 50 learning trials:

Which Algorithm?
Q-learning, no features, 1000 learning trials:

Which Algorithm?
Q-learning, simple features, 50 learning trials:

Partially observable MDPs

- Markov decision processes:
  - States \( S \)
  - Actions \( A \)
  - Transitions \( P(s'|s,a) \) (or \( T(s,a,s') \))
  - Rewards \( R(s,a,s') \) (and discount \( \gamma \))
  - Start state distribution \( b_0 = P(s_0) \)

- POMDPs, just add:
  - Observations \( O \)
  - Observation model \( P(o|s,a) \) (or \( Q(s,a,o) \))

A POMDP: Ghost Hunter
POMDP Computations

- Sufficient statistic: belief states
  - $b_i = \Pr(s_i)$
- POMDPs search trees
  - max nodes are belief states
  - expectation nodes branch on possible observations
  - (this is motivational; we will not discuss in detail)

Types of Planning Problems

<table>
<thead>
<tr>
<th>State</th>
<th>Action Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical Planning</td>
<td>observable</td>
</tr>
<tr>
<td>MDPs</td>
<td>observable</td>
</tr>
<tr>
<td>POMDPs</td>
<td>partially observable</td>
</tr>
</tbody>
</table>

Classical Planning

- World deterministic
- State observable

MDP-Style Planning

- Policy
- Universal Plan
- Navigation function
- World stochastic
- State observable

Stochastic, Partially Observable

- Sign: 50% 50%
- Start: heaven, hell

Stochastic, Partially Observable

- Heaven, Hell
- Start: heaven, hell
- Sign: 50% 50%
Notation (1)

- Recall the Bellman optimality equation:
  \[ V'(s) = \max_{a \in A(s)} \left[ R'_s + \gamma V'(s') \right] \]
- Throughout this section we assume
  \[ R'_s = \frac{1}{\gamma} R_s = \frac{1}{\gamma} r(s, a) \]
  is independent of \( s' \) so that the Bellman optimality equation turns into
  \[ V'(s) = \gamma \max_{a \in A(s)} \left[ R'_s + \sum_{s'} V'(s') P_{s's} \right] = \gamma \max_{a \in A(s)} \left[ r(s, a) + \sum_{s'} V'(s') P_{s's} \right] \]

Value Iteration

- Given this notation the value iteration formula is
  \[ V_T(x) = \gamma \max_{u} \left[ r(x, u) + \int V_{T-1}(x') P(x' | u, x) \, dx' \right] \]
  with
  \[ V_1(b) = \gamma \max_{u} r(x, u) \]

Notation (2)

- In the remainder we will use a slightly different notation for this equation:
  \[ V(x) = \max_{u} \left[ r(x, u) + \int V(x') P(x' | u, x) \, dx' \right] \]
- According to the previously used notation we would write
  \[ V'(s) = \gamma \max_{a \in A(s)} \left[ r(s, a) + \sum_{s'} V'(s') P_{s's} \right] \]
- We replaced \( x \) by \( s \) and \( u \) by \( a \), and turned the sum into an integral.

POMDPs

- In POMDPs we apply the very same idea as in MDPs.
- Since the state is not observable, the agent has to make its decisions based on the belief state which is a posterior distribution over states.
- Let \( b \) be the belief of the agent about the state under consideration.
- POMDPs compute a value function over belief spaces:
  \[ V_T(b) = \gamma \max_{u} \left[ r(b, u) + \int V_{T-1}(b') P(b' | u, b) \, db' \right] \]
Problems

- Each belief is a probability distribution, thus, each value in a POMDP is a function of an entire probability distribution.
- This is problematic, since probability distributions are continuous.
- Additionally, we have to deal with the huge complexity of belief spaces.
- For finite worlds with finite state, action, and measurement spaces and finite horizons, however, we can effectively represent the value functions by piecewise linear functions.

The Parameters of the Example

- The actions $u_1$ and $u_2$ are terminal actions.
- The action $u_3$ is a sensing action that potentially leads to a state transition.
- The horizon is finite and $\gamma=1$.

Payoffs in Our Example (1)

- If we are totally certain that we are in state $x_1$ and execute action $u_1$, we receive a reward of -100.
- If, on the other hand, we definitely know that we are in $x_2$ and execute $u_1$, the reward is +100.
- In between it is the linear combination of the extreme values weighted by their probabilities

Payoffs in Our Example (2)

- In MDPs, the payoff (or return) depended on the state of the system.
- In POMDPs, however, the true state is not exactly known.
- Therefore, we compute the expected payoff by integrating over all states:

Payoff in POMDPs

- In MDPs, the payoff (or return) depended on the state of the system.
- In POMDPs, however, the true state is not exactly known.
- Therefore, we compute the expected payoff by integrating over all states:

$$r(b, u) = \mathbb{E}_x[r(x, u)] = \int r(x, u)p(x) \, dx = p_1 \cdot r(x_1, u_1) + p_2 \cdot r(x_2, u_2)$$
The Resulting Policy for $T=1$
- Given we have a finite POMDP with $T=1$, we would use $V_1(b)$ to determine the optimal policy.
- In our example, the optimal policy for $T=1$ is
  \[
  \pi_1(b) = \begin{cases} 
  u_1 & \text{if } p_1 \leq \frac{3}{7} \\
  u_2 & \text{if } p_1 > \frac{3}{7}
  \end{cases}
  \]
- This is the upper thick graph in the diagram.

Piecewise Linearity, Convexity
- The resulting value function $V_j(b)$ is the maximum of the three functions at each point
  \[
  V_1(b) = \max_u r(b, u) = \max \left\{ \begin{array}{l}
  -100 p_1 + 100 (1 - p_1) \\
  100 p_1 - 50 (1 - p_1)
  \end{array} \right. 
  \]
- It is piecewise linear and convex.

Pruning
- If we carefully consider $V_j(b)$, we see that only the first two components contribute.
- The third component can therefore safely be pruned away from $V_j(b)$.

Increasing the Time Horizon
- If we go over to a time horizon of $T=2$, the agent can also consider the sensing action $u_3$.
- Suppose we perceive $z$, for which $p(z_1 | x_1) = 0.7$ and $p(z_1 | x_2) = 0.3$.
- Given the observation $z$, we update the belief using Bayes rule.
- Thus $V_j(b | z)$ is given by
  \[
  V_j(b | z_1) = \max \left\{ \begin{array}{l}
  -100 \cdot \frac{0.7}{p(z_1)} + 100 \cdot \frac{0.3}{p(z_1)} \\
  100 \cdot \frac{0.7}{p(z_1)} - 50 \cdot \frac{0.3}{p(z_1)}
  \end{array} \right. 
  \]

Expected Value after Measuring
- Since we do not know in advance what the next measurement will be, we have to compute the expected belief
  \[
  \bar{V}_1(b) = \mathbb{E}_z[V_1(b | z)] = \max \left\{ \begin{array}{l}
  -70 p_1 + 30 (1 - p_1) \\
  70 p_1 - 15 (1 - p_1)
  \end{array} \right. 
  \]

Resulting Value Function
- The four possible combinations yield the following function which again can be simplified and pruned.
  \[
  \bar{V}_2(b) = \max \left\{ \begin{array}{l}
  -70 p_1 + 30 (1 - p_1) - 30 p_1 + 70 (1 - p_1) \\
  -70 p_1 + 30 (1 - p_1) + 30 p_1 - 35 (1 - p_1) \\
  +70 p_1 - 15 (1 - p_1) - 30 p_1 + 70 (1 - p_1) \\
  +70 p_1 - 15 (1 - p_1) + 30 p_1 - 35 (1 - p_1)
  \end{array} \right. 
  \]

\[
= \max \left\{ \begin{array}{l}
  -100 p_1 + 100 (1 - p_1) \\
  +40 p_1 + 55 (1 - p_1) \\
  +100 p_1 - 50 (1 - p_1)
  \end{array} \right. 
  \]
State Transitions (Prediction)

- When the agent selects \( u_3 \), its state potentially changes.
- When computing the value function, we have to take these potential state changes into account.

\[
p'_1 = E_x [p(x_1 | x, u_3)] \\
= \sum_{i=1}^{2} p(x_1 | x_i, u_3)p_i \\
= 0.2p_1 + 0.8(1 - p_1) \\
= 0.8 - 0.6p_1
\]

Resulting Value Function after executing \( u_3 \)

- Taking also the state transitions into account, we finally obtain.

\[
\hat{V}_1(b | u_3) = \max \left\{ \begin{array}{l}
60p_1 - 60(1 - p_1) \\
52p_1 + 43(1 - p_1) \\
-20p_1 + 70(1 - p_1)
\end{array} \right\}
\]

Value Function for T=2

- Taking into account that the agent can either directly perform \( u_1 \) or \( u_2 \), or first \( u_3 \) and then \( u_1 \) or \( u_2 \), we obtain (after pruning)

\[
\hat{V}_2(b) = \max \left\{ \begin{array}{l}
-100p_1 + 100(1 - p_1) \\
100p_1 - 50(1 - p_1) \\
51p_1 + 42(1 - p_1)
\end{array} \right\}
\]

Deep Horizons and Pruning

- We have now completed a full backup in belief space.
- This process can be applied recursively.
- The value functions for T=10 and T=20 are

Why Pruning is Essential

- Each update introduces additional linear components to \( V \).
- Each measurement squares the number of linear components.
- Thus, an unpruned value function for T=20 includes more than \( 10^{547,864} \) linear functions.
- At T=30 we have \( 10^{561,012,337} \) linear functions.
- The pruned value functions at T=20, in comparison, contains only 12 linear components.
- The combinatorial explosion of linear components in the value function are the major reason why POMDPs are impractical for most applications.
A Summary on POMDPs

- POMDPs compute the optimal action in partially observable, stochastic domains.
- For finite horizon problems, the resulting value functions are piecewise linear and convex.
- In each iteration the number of linear constraints grows exponentially.
- POMDPs so far have only been applied successfully to very small state spaces with small numbers of possible observations and actions.