Recap: MDPs

Markov decision processes:
- States $S$
- Start state $s_0$
- Actions $A$
- Transition $p(s'|s,a)$ (or $T(s,a,s')$)
- Reward $R(s,a,s')$ (and discount $\gamma$)

MDP quantities:
- Policy = Choice of action for each (MAX) state
- Utility (or return) = sum of discounted rewards
Optimal Utilities

- The value of a state $s$:
  - $V^*(s) =$ expected utility starting in $s$ and acting optimally

- The value of a Q-state $(s,a)$:
  - $Q^*(s,a) =$ expected utility starting in $s$, taking action $a$ and thereafter acting optimally

- The optimal policy:
  - $\pi^*(s) =$ optimal action from state $s$
Solving MDPs

➢ Value iteration
  • Start with $V_1(s) = 0$
  • Given $V_i$, calculate the values for all states for depth $i+1$:

\[
V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i(s')] \\
\]
  • Repeat until converge
  • Use $V_i$ as evaluation function when computing $V_{i+1}$

➢ Policy iteration
  • Step 1: policy evaluation: calculate utilities for some fixed policy
  • Step 2: policy improvement: update policy using one-step look-ahead with resulting utilities as future values
  • Repeat until policy converges
Reinforcement learning

- Don’t know T and/or R, but can observe R
  - Learn by doing
  - can have multiple trials
The Story So Far: MDPs and RL

Things we know how to do:

- If we know the MDP
  - Compute $V^*$, $Q^*$, $\pi^*$ exactly
  - Evaluate a fixed policy $\pi$

- If we don’t know $T$ and $R$
  - If we can estimate the MDP then solve
  - We can estimate $V$ for a fixed policy $\pi$
  - We can estimate $Q^*(s,a)$ for the optimal policy while executing an exploration policy

Techniques:

- Computation
  - Value and policy iteration
  - Policy evaluation

- Model-based RL
  - sampling

- Model-free RL:
  - Q-learning
Model-Free Learning

- Model-free (temporal difference) learning
  - Experience world through trials 
    \((s, a, r, s', a', r', s'', a'', r'', s''' \ldots)\)
  - Update estimates each transition \((s, a, r, s')\)
  - Over time, updates will mimic Bellman updates

Q-Value Iteration (model-based, requires known MDP)

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

Q-Learning (model-free, requires only experienced transitions)

\[
Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') \right]
\]
Q-Learning

- Q-learning produces tables of q-values:
Exploration / Exploitation

Random actions ($\varepsilon$ greedy)
- Every time step, flip a coin
- With probability $\varepsilon$, act randomly
- With probability $1-\varepsilon$, act according to current policy
Today: Q-Learning with state abstraction

- In realistic situations, we cannot 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 want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we’ll see it over and over again
Example: Pacman

- Let's say we discover through experience that this state is bad:

- In naive Q-learning, we know nothing about this state or its Q-states:

- Or even this 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/(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)

Similar to a evaluation function
Linear Feature Functions

- Using a feature representation, we can write a Q function for any state using a few weights:

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

- Advantage: more efficient learning from samples
- Disadvantage: states may share features but actually be very different in value!
Function Approximation

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

- \( Q \)-learning with linear \( Q \)-functions:
  - transition = (s,a,r,s')
    - difference = \[ r + \gamma \max_{a'} Q(s',a') \] - \( Q(s,a) \)
    - \( Q(s,a) \leftarrow Q(s,a) + \alpha \text{[difference]} \)
    - \( w_i \leftarrow w_i + \alpha \text{[difference]} f_i(s,a) \)

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state’s features

Exact Q’s
Approximate Q’s
Example: Q-Pacman

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

\[ f_{\text{DOT}}(s, \text{NORTH}) = 0.5 \]
\[ f_{\text{GST}}(s, \text{NORTH}) = 1.0 \]

\[ Q(s,a) = +1 \]

\[ R(s,a,s') = -500 \]
\[ \text{difference} = -501 \]

\[ w_{\text{DOT}} \leftarrow 4.0 + \alpha [-501] 0.5 \]
\[ w_{\text{GST}} \leftarrow -1.0 + \alpha [-501] 1.0 \]

\[ Q(s,a) = 3.0 f_{\text{DOT}}(s,a) - 3.0 f_{\text{GST}}(s,a) \]
Linear Regression

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

\[ y = w_0 + w_1 f_1(x) + w_2 f_2(x) \]
Ordinary Least Squares (OLS)

\[
\text{total error} = \sum_i \left(y_i - \hat{y}_i\right)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x)\right)^2
\]
Minimizing Error

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

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

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

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

Approximate $q$ update as a one-step gradient descent:

\[
w_m \leftarrow w_m + \alpha \left[ r + \gamma \max_a Q(s', a') - Q(s, a) \right] f_m(x)
\]
How many features should we use?

- As many as possible?
  - computational burden
  - overfitting

- Feature selection is important
  - requires domain expertise
Overfitting

Degree 15 polynomial
Overview of Project 3

- MDPs
  - Q1: value iteration
  - Q2: find parameters that lead to certain optimal policy
  - Q3: similar to Q2

- Q-learning
  - Q4: implement the Q-learning algorithm
  - Q5: implement $\epsilon$ greedy action selection
  - Q6: try the algorithm

- Approximate Q-learning and state abstraction
  - Q7: Pacman

- Tips
  - make your implementation general