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 the MDP
  - 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:

- Model-based DPs
  - Value Iteration
  - Policy evaluation

- Model-based RL

- Model-free RL
  - Value learning
  - Q-learning

This slide deck courtesy of Dan Klein at UC Berkeley
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:
    \[
    Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right]
    \]
    \[
    sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')
    \]
  - Incorporate the new estimate into a running average:
    \[
    Q(s,a) \leftarrow (1 - \alpha)Q(s,a) + (\alpha) [sample]
    \]
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!
  - Basically doesn’t matter how you select actions (!)

- Neat property: off-policy learning
  - learn optimal policy without following it (some caveats)
Exploration / Exploitation

- Several schemes for forcing exploration
  - 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

- **Q-learning produces tables of q-values:**

![Q-Values After 1000 Episodes](image)

- **Q-VALUES AFTER 1000 EPISODES**
Q-Learning

- 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 naïve 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
  - 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 few weights:

\[ V(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

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

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but 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:**

\[
Q(s, a) \leftarrow Q(s, a) + \alpha [\text{error}]
\]

\[
w_i \leftarrow w_i + \alpha [\text{error}] f_i(s, a)
\]

- **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

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

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

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

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

\[ R(s, a, s') = -500 \]

\[ \text{error} = -501 \]

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

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

\[ Q(s, a) = 3.0 f_{\text{DOT}}(s, a) - 3.0 f_{\text{GST}}(s, a) \]
Given examples \( (x_i, y_i)_{i=1}^{n} \)
Predict \( y_{n+1} \) given a new point \( x_{n+1} \)
Linear regression

Prediction
\[ \hat{y}_i = w_0 + w_1 x_i \]

Prediction
\[ \hat{y}_i = w_0 + w_1 x_{i,1} + w_2 x_{i,2} \]
Ordinary Least Squares (OLS)

Observation $y$
Prediction $\hat{y}$

Error or “residual”

$$\sum_{i} \left( \sum_{k} f_k(x_i) w_k - y_i \right)^2$$
Minimizing Error

\[ E(w) = \frac{1}{2} \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right)^2 \]

\[ \frac{\partial E}{\partial w_m} = \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right) f_m(x_i) \]

\[ E \leftarrow E + \alpha \sum_i \left( \sum_k f_k(x_i) w_k - y_i \right) f_m(x_i) \]

Value update explained:

\[ w_i \leftarrow w_i + \alpha \text{[error]} f_i(s, a) \]
Overfitting

Degree 15 polynomial
Policy Search
Policy Search

- Problem: often the feature-based policies that work well aren’t the ones that approximate V / Q best
  - E.g. your value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions
  - We’ll see this distinction between modeling and prediction again later in the course

- Solution: learn the policy that maximizes rewards rather than the value that predicts rewards

- This is the idea behind policy search, such as what controlled the upside-down helicopter
Policy Search

- Simplest policy search:
  - Start with an initial linear value function or q-function
  - Nudge each feature weight up and down and see if your policy is better than before

- Problems:
  - How do we tell the policy got better?
  - Need to run many sample episodes!
  - If there are a lot of features, this can be impractical
Policy Search*

- Advanced policy search:
  - Write a stochastic (soft) policy:
    \[
    \pi_w(s) \propto e^{\sum_i w_i f_i(s,a)}
    \]
  - Turns out you can efficiently approximate the derivative of the returns with respect to the parameters \(w\) (details in the book, but you don’t have to know them)
  - Take uphill steps, recalculate derivatives, etc.
Take a Deep Breath…

- We’re done with search and MDPs!

- Next, we’ll look at how to reason with probabilities
  - Diagnosis
  - Tracking objects
  - … lots more!

- Last part of course: machine learning, classical planning
A (Short) History of AI

- **1940-1950: Early days**
  - 1943: McCulloch & Pitts: Boolean circuit model of brain
  - 1950: Turing’s “Computing Machinery and Intelligence”

- **1950—70: Excitement: Look, Ma, no hands!**
  - 1950s: Early AI programs, including Samuel's checkers program, Newell & Simon's Logic Theorist, Gelernter's Geometry Engine
  - 1956: Dartmouth meeting: “Artificial Intelligence” adopted
  - 1965: Robinson's complete algorithm for logical reasoning

- **1970—88: Knowledge-based approaches**
  - 1969—79: Early development of knowledge-based systems
  - 1980—88: Expert systems industry booms

- **1988—: Statistical approaches**
  - Resurgence of probability, focus on uncertainty
  - General increase in technical depth
  - Agents and learning systems… “AI Spring”?

- **2000—: Where are we now?**