Reinforcement Learning

- We still assume an MDP:
  - A set of states $s \in S$
  - A set of actions (per state) $A$
  - A model $T(s, a, s')$
  - A reward function $R(s, a, s')$
  - Still looking for a policy $\pi(s)$

- New twist: don’t know $T$ or $R$, so must try out actions

- Big idea: Compute all averages over $T$ using sample outcomes

Approximating Values through Samples

- Policy Evaluation:
  $$v_{\pi+1}(s) = \sum_s T(s, \tau(s), s')[R(s, \tau(s), s') + \gamma v_{\pi}(s')]$$

- Value Iteration:
  $$v_{\pi+1}(s) = \max_a \sum_s T(s, a, s') [R(s, a, s') + \gamma v_{\pi}(s')]$$

- Q-Value Iteration:
  $$Q_{\pi+1}(s, a) = \sum_s T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_{\pi}(s', a')]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration
  $$Q_{\pi+1}(s, a) = \sum_s T(s, a, s') [R(s, a, s') + \gamma \max_{a'} Q_{\pi}(s', a')]$$

- Learn $Q(s, a)$ values as you go
  - Receive a sample $(s, a, s', r)$
  - Consider your old estimate $Q_{\pi}(s, a)$
  - Consider your new estimate with sample $s' = (1 - \alpha) Q_{\pi}(s, a) + \alpha$ [sample]

- Incorporate the new estimate into a running average:
  $$Q(s, a) = (1 - \alpha) Q(s, a) + \alpha \text{ [sample]}$$

Exploration vs. Exploitation

- Exploring the Maze
  - Exploration
  - Value Learning
  - Q-Learning
  - Value Iteration

The Story So Far: MDPs and RL

- Known MDP: Offline Solution
  - Goal: Compute $V^*, Q^*, \pi^*$
  - Technique: Value / Policy Iteration

- Unknown MDP: Model-Based
  - Goal: Compute $V^*, Q^*, \pi^*$
  - Technique: Value Learning

- Unknown MDP: Model-Free
  - Goal: Compute $V^*, Q^*, \pi^*$
  - Technique: Q-Learning
**Exploration Functions**

- **When to explore?**
  - Random actions: explore a fixed amount.
  - Better idea: explore areas whose badness is not (yet) established, eventually stop exploring.

- **Exploration function**
  - Takes a value estimate $q(s,a)$ and a visit count $n$, and
  - Returns an optimistic utility, e.g. $q(s,a) + \frac{1}{\sqrt{n}}$.
  - **Regular Q-Update**: $Q(s,a) \leftarrow R(s,a,s') + \gamma Q(s',a')$
  - **Modified Q-Update**: $Q(s,a) \leftarrow R(s,a,s') + \gamma \min_a Q(s',a') + \frac{1}{n(s,a)}$

- **Note:** this propagates the “bonus” back to states that lead to unknown states as well.

**Regret**

- Even if you learn the optimal policy, you still make mistakes along the way!
- Regret is a measure of your total mistake cost: the difference between your (expected) rewards, including youthful suboptimality, and optimal (expected) rewards.
- Minimizing regret goes beyond learning to be optimal: it requires optimally learning to be optimal.
- Example: random exploration and exploration functions both end up optimal, but random exploration has higher regret.

**Approximate Q-Learning**

**Generalizing Across States**

- Basic Q-Learning keeps a table of all $q$-values.
- 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.
  - Generalize that experience to new, similar situations.
  - 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 even this one?

**Video of Demo Q-Learning Pacman – Tiny – Watch All**
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
    - Number of dots in sight
    - Number of ghosts
    - 1 (if this is a ghost)
    - Is the state visible to Pacman?
    - Is Pacman in a tunnel? (0/1)
    - Is this the exact state on the slide?
    - Example features: distance to closest dot, number of ghosts, 1/ distance to dot, etc.
- Can also describe a q-state (s, a) with features (e.g., which moves closer to food).

Linear Value 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 actually be very different in value.

Example: Q-Pacman

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

Approximate Q-Learning

- Q-learning with linear Q-functions:
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \ldots + w_n f_n(s, a) \]
- Intuitive interpretation:
  - Adjust weights of active features.
  - E.g., if something unexpectedly bad happens, blame the features that were active during that state.
- Formal justification: online least squares.
DeepMind Atari (© Two Minute Lectures)
approximate Q-learning with neural nets

Q-Learning and Least Squares

Linear Approximation: Regression

Optimization: Least Squares

Minimizing Error

More Powerful Function Approximation

- Linear:
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \cdots + w_n f_n(s, a) \]
- Polynomial:
  \[ Q(s, a) = w_{11} f_1(s, a) + w_{12} f_2(s, a)^2 + \cdots + w_{1n} f_n(s, a)^n \]
- Neural Network:
  \[ Q(s, a) = w_1 f_1(s, a) + w_2 f_2(s, a) + \cdots + w_n f_n(s, a) \]
More Powerful Function Approximation

- Summary of iterative feature update equation!

\[ w_{n+1} = w_n + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right) \frac{dQ}{dw_{n+1}}(s, a) \]

= \[ f_m(s, a) \] when linear approximation

Overfitting: Why Limiting Capacity Can Help

RL: Learning Soccer

[Bansal et al, 2017]

RL: Learning Manipulation

[Levine*, Finn*, Darrell, Abbeel, JMLR 2016]

RL: NASA SUPERball

Problem: often the feature-based policies that work well (win games, maximize utilities) aren't the ones that approximate $V$ / $Q$ best.

- E.g., our value functions from project 2 were probably horrible estimates of future rewards, but they still produced good decisions.
- Q-learning's priority: get $Q$ values close (modeling)
- Action-selection priority: get ordering of $Q$ values right (prediction)
- We'll see this distinction between modeling and prediction again later in the course.

Solution: learn policies that maximize rewards, not the values that predict them.

Policy search: start with an ok solution (e.g., Q-learning) then fine-tune by hill climbing on feature weights.

- 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

- Better methods exploit lookahead structure, sample wisely, change multiple parameters, ...
Advanced Topic: Q-learning for Atari

Key implementations:
- Neural network approximation of Q-values
- Handling large state-action space (state compressing function)
- Experience Replay!

Experience Replay: idea in RL where you re-select past transitions or episodes to repeatedly train on them.

Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning.
- Next up: Part II: Uncertainty and Learning!