Artificial Intelligence Programming

Reinforcement Learning

Cindi Thompson

Department of Computer Science

University of San Francisco

Previously ...

- Tuesday we saw how to solve a Markov Decision Process.
- This gives us an optimal policy
  - Mapping of states to actions
  - Lets our agent act optimally (in expectation) in stochastic environments.
- Value Iteration and Policy Iteration assume a lot of available knowledge.
  - All rewards, all state transitions
- Can we learn the values of states without this information?

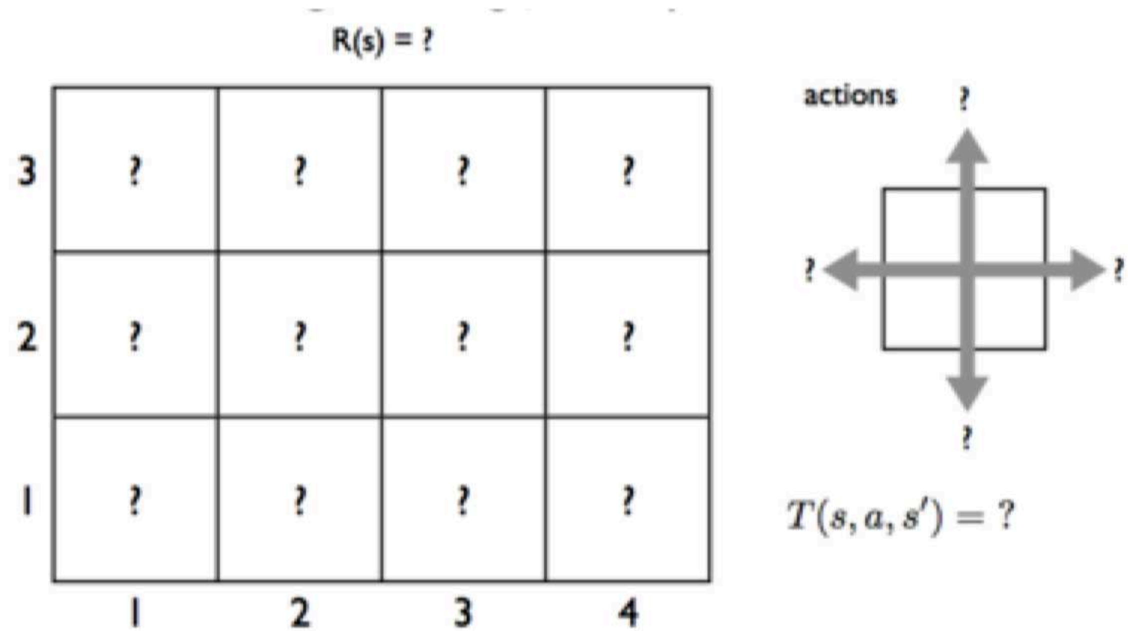
Reinforcement Learning

- So far, most of the learning algorithms we've looked at have been *supervised*, *passive*, and *offline*.
  - We're given a labeled dataset and need to construct a hypothesis.
- Sometimes our agent is not that lucky.
  - Placed in an environment and forced to learn the best action to take in each state.
  - The environment provides a reward to the agent, but never tells it the best thing to do.
  - The agent must select actions to take.

Reinforcement Learning

- This approach to learning is called *reinforcement learning*.
- Not too different from the way children or animals are trained.
    - Reward positive actions
    - Punish negative actions

Reinforcement Learning Scenario



RL Scenario

- Don’t know anything! Unknown environment, transition model, and reward function
- Need to explore the world. General approach: at each state the agent:
  - Selects an available action (it at least knows that!)
  - Receives reward (it can still sense rewards)
  - Observes resulting state (can still sense current state)
  - Repeat until a terminal state

Goal: find an optimal policy from these observations

Reinforcement Learning

Let's start simpler

- How could we do this in a deterministic, episodic environment?
  - Each action leads to a single reward.

Reinforcement Learning

- How could we do this in a deterministic, episodic environment?
  - Each action leads to a single reward.
  - Try each action once, then always select the action with the highest reward.

Reinforcement Learning

- How could we do this in a deterministic, sequential environment?
- Case 1:
  - A sequence of actions leads to a reward at the end.
  - Try each sequence once, then always select the sequence with the highest reward.
- Case 2:
  - Each action has an immediate reward.
  - Use search to find the optimal sequence of actions.

Reinforcement Learning

- What about in a stochastic, episodic environment?
  - Each action has multiple possible outcomes.

Reinforcement Learning

- What about in a stochastic, episodic environment?
  - Each action has multiple possible outcomes.
  - We try each action multiple times and learn the expected utility of each action.
    - This is sometimes called a *bandit problem*

Reinforcement Learning

- What about in a stochastic, episodic environment?
  - Each action has multiple possible outcomes.
  - We try each action multiple times and learn the expected utility of each action.
    - How many times is “enough”?
    - Often, our agent is doing *active learning*; must integrate learning and performance.

Exploration

- Issue: The agent would always like to choose the action with the highest reward.
- This means taking the action with the highest expected utility.
- But if the agent never tries 'bad-looking' actions, it might be missing out on a higher reward.
  - Example: consider two levers:
    - one pays $1 every time, and
    - the other pays $0 90% of the time, and $100 10% of the time.

Exploration

- Issue: early on, we would like to try lots of "non-optimal" actions, as our estimates are probably very inaccurate.
  - This process is called *exploration*.
- As our estimates of the reward for each action get more accurate, we want to select the best action more frequently.
  - This process is called *exploitation*.
- How to do this in a principled way?

Boltzmann exploration

- One way to do this is using Boltzmann exploration.
- Let $U'(a)$ be our estimated utility for taking action a.
- We take an action with probability:

$$\frac{exp(\frac{U(a)}{k})}{\sum_a exp(\frac{U(a)}{k})}$$

- Where k is a temperature parameter.
  - k starts high and gradually decreases.
- How will this behave with $k = 1$? $k << 1$?

Sequential Problems

- This works great for episodic problems, but what about sequential problems?
- Now we need to think not only about the immediate reward an action generates, but also the value of subsequent states.
- How did we solve this with full information?

Markov Decision Process

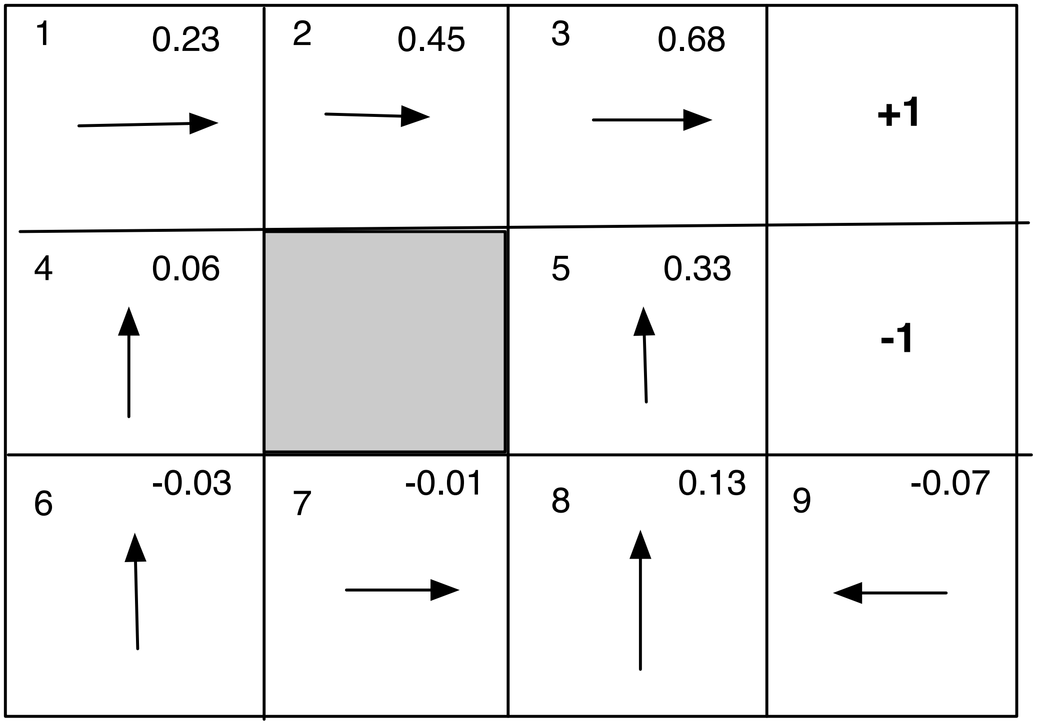
- With full information, we modeled this problem as a Markov Decision Process.
- We knew all the states, the transition probabilities between each state, and the rewards for each state.
- We then used value iteration to estimate utilities for each state, or policy iteration to find the policy directly.

Example Utilities

3	0.812	0.868	0.918	+ 1
2	0.762		0.660	–1
1	0.705	0.655	0.611	0.388
	1	2	3	4

- Value iteration can find utilities for each state, and we use these to construct a policy.

Example Policies



- Policy iteration lets us find policies directly.

Reinforcement learning

- Both value iteration and policy iteration assume a great deal of knowledge about the world.
  - Reward function
  - Transition function
- What if we don't have a model?
- All we know is that there are a set of states, and a set of actions.

Model Estimation

We don't even know the transition function! P(s'|s,a). Might approximate by:

$$\frac{\#\texttt{transitions} s \rightarrow s' \texttt{for} a}{\#\texttt{times}}$$

- For example, we tried "up" from (1,1) 10 times, and we observe:
  - 8 times we get to (1,2) and R=-.04
  - 2 times we get to (2,1) and R=-.04
- Now P((1,2)| (1,1),"up") = 8/10
- and R(s) for both states = -.04

Continued exploration will give increasingly accurate estimates. Do value / policy iteration as before.

Model-free learning

- This is pretty slow!
- We really want to learn an optimal *policy*, don't really care about utilities.
- This is called model-free learning: don't even have to learn transition function

Q-learning

- Learning a policy directly is difficult.
- Problem: our data is not of the form: <state, action>
- Instead, it's of the form $s_1, s_2, s_3, ..., R$.
- Since we don't know the transition function, it's also hard to learn the utility of a state.
- Instead, we'll learn a function $Q(s, a)$. This will estimate the "utility" of taking action a in state s.

Q-learning

- More precisely, $Q(s, a)$ will represent the value of taking a in state s, then acting optimally after that.
- $Q(s, a) = R(s, a) + \gamma max_a \sum_{s'} P(s'|s, a)U(s')$
  - γ is our discount factor
- The optimal policy is then to take the action with the highest Q value in each state.
- Of course, we don't know what transition probabilities or utilities are.

Learning the Q function

- To learn Q, we need to be able to estimate the value of taking an action in a state even though our rewards are spread out over time.
- We can do this iteratively.
- Notice that $U(s) = max_a Q(s, a)$
- We can then rewrite our equation for Q as:
- $Q(s, a) = R(s, a) + \gamma max_{a'} Q(s', a')$

Learning the Q function

- Let's denote our estimate of $Q(s,a)$ as $\hat{Q}(s,a)$
- We'll keep a table listing each state-action pair and estimated Q-value
- the agent observes its state s, chooses an action a, then observes the reward $r = R(s,a)$ that it receives and the new state s'.
- In the simplest case, it then updates the Q-table according to the following formula:

$$\hat{Q}(s,a) = r + \gamma max_{a'} \hat{Q}(s',a')$$

Learning the Q function

- The agent uses the estimate of $\hat{Q}$ for s' to estimate $\hat{Q}$ for s.
- Notice that the agent doesn't need any knowledge of R or the transition function to execute this.
- But why put so much reliance on what just happened?
- so we introduce a *learning rate* α that trades off between what we have already learned and the action just taken. Lower over time:

$$\hat{Q}(s,a) = (1-\alpha) \times \hat{Q}(s,a) + \alpha \times (r + \gamma max_{a'} \hat{Q}(s',a'))$$

Convergence

- Q-learning is guaranteed to converge as long as:
  - Rewards are bounded
  - The agent selects state-action pairs in such a way that it sees each infinitely often.
  - This means that an agent must have a nonzero probability of selecting each a in each s as the sequence of state-action pairs approaches infinity.

Exploration

- Of course, these are theoretical guarantees.
- In practice, we must sample each state “enough”.
- We can do this with Boltzmann exploration.

$$\frac{exp(\frac{\hat{Q}(s,a)}{k})}{\sum_a exp(\frac{\hat{Q}(s,a)}{k})}$$

- In the next slide, we use “bestAction(s2)” as an abbreviation of $max_{a'}\hat{Q}(s', a')$

Pseudocode

```
Initialize Q table randomly
s = randomState
while not done :
  a = selectAction(s) # use Boltzmann here
  s2 = takeAction(s,a)
  r = reward(s2)
  Q(s,a) = (1-alpha)*Q(s,a) + alpha * (reward +gamma * bestAction(s2))
  if s2 is goal
     s = randomState
  else
     s = s2
```

Pros and cons

- Q-learning has proved to be very useful in some settings.
  - No knowledge of problem dynamics needed.
  - No labeled training data needed.
  - Agent can integrate learning and execution.
- It also has some weaknesses
  - Convergence can be slow
  - Q-table is very large
  - Not much generalization

Incorporating generalization

- A big weakness of Q-learning as compared to other learning algorithms is that knowing what to do in state s tells me nothing about how to act in states that are very similar to s.
  - Q-learning has poor *generalization*
  - This is a result of storing all the state-action pairs as a table.
- A standard way to deal with this is to store the table in a *function approximator*.
  - A construct that maps $< state, action >$ to reward.

Credit Assignment

- Q-learning works best in environments in which reward is immediate.
- In delayed-reward environments, the *credit assignment problem* becomes an issue.
- To address this, a more sophisticated method known as *TD-learning* can be used.
  - Idea - look ahead to the best action in the next state and use its EU.

Summary

- Policies tell our agent how to act optimally in a stochastic world.
- Reinforcement learning lets us learn a Q function, which gives us a policy.
- No knowledge of the world needed - we just have to be able to select actions and observe rewards.
- This is active, online learning.
- Most extreme example of trading knowledge for time to learn.