Review: MDPs

S = set of states set (|S| = n)

A = set of actions (|A| = m)

Pr = transition function Pr(s,a,s')

represented by set of m n x n stochastic matrices
each defines a distribution over S x S

R(s) = bounded, real-valued reward function
represented by an n-vector

Goal for an MDP

• Find a policy which:
  maximizes expected discounted reward
  over an infinite horizon
  for a fully observable
  Markov decision process.
Bellman Backup

\[ Q_{n+1}(s,a) = V_n(s), \ V_n(s'), \ V_n(s), \ V_n(s), \ ... \]

Improve estimate of value function

\[ V_{t+1}(s) = R(s) + \max_{a \in A} \left\{ c(a) + \gamma \sum_{s'} \mathbb{P}(s'|a,s) V_t(s') \right\} \]

Expected future reward
Avergd over dest states

Value Iteration

• Assign arbitrary values to each state
(or use an admissible heuristic).

• Iterate over all states
Improving value funct via Bellman Backups

• Stop the iteration when converges
\( (V_t \text{ approaches } V^* \text{ as } t \to \infty) \)

• Dynamic Programming

How is learning to act possible when...

• Actions have non-deterministic effects
  Which are initially unknown

• Rewards / punishments are infrequent
  Often at the end of long sequences of actions

• Learner must decide what actions to take

• World is large and complex

Naïve Approach

1. Act Randomly for a while
   (Or systematically explore all possible actions)

2. Learn
   Transition function
   Reward function

3. Use value iteration, policy iteration, ...

Problems?
Example:
- Suppose given policy
- Want to determine how good it is

Objective: Value Function

Passive RL
- Given policy $\pi$, estimate $U(\pi)(s)$
- Not given transition matrix, nor reward function!
- Epochs: training sequences

Approach 1
- Direct estimation
Estimate $U(s)$ as average total reward of epochs containing $s$ (calculating from $s$ to end of epoch)
- Pros / Cons?

Requires huge amount of data
doesn’t exploit Bellman constraints!

Expected utility of a state =
its own reward +
expected utility of successors
Temporal Difference Learning

Do backups on a per-action basis
Don’t try to estimate entire transition function!
For each transition from $s$ to $s'$, update:

\[
\alpha = \text{Learning rate} \\
U^\pi(s) \leftarrow U^\pi(s) + \alpha \left( R(s) + \gamma U^\pi(s') - U^\pi(s) \right)
\]

$\gamma = \text{Discount rate}$

Notes

• Once $U$ is learned, updates become 0:
  \[
  \text{when } U^\pi(s) = R(s) + \gamma U^\pi(s')
  \]

Adjusts state to ‘agree’ with observed successor

• Not all possible successors

Doesn’t require $M$, model of transition function

Notes II

• “TD(0)"
  One step lookahead
  \[
  U^\pi(s) \leftarrow U^\pi(s) + \alpha \left( R(s) + \gamma U^\pi(s') - U^\pi(s) \right)
  \]

Can do 2 step, 3 step...

TD(\lambda)

• Or, ... take it to the limit!
• Compute weighted average of all future states

\[
U^\pi(s_i) \leftarrow U^\pi(s_i) + \alpha \left( R(s_i) + \gamma U^\pi(s_{i+1}) - U^\pi(s_i) \right)
\]

becomes

\[
U^\pi(s_i) \leftarrow U^\pi(s_i) + \alpha \left( R(s_i) + \gamma (1 - \lambda) \sum_{i=1}^{\infty} \lambda^i U^\pi(s_{i+1}) - U^\pi(s_i) \right)
\]

• Implementation
  Propagate current weighted TD onto past states
  Must memorize states visited from start of epoch
Q-Learning

- Version of TD-learning where instead of learning value function on states we learn function on \([\text{state, action}]\) pairs

\[
U^\pi(s) \leftarrow U^\pi(s) + \alpha (R(s) + \gamma \max_a Q(s', a) - U^\pi(s))
\]

becomes

\[
Q(s, a) \leftarrow Q(s, a) + \alpha [R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)]
\]

- [Helpful for model-free policy learning]

Active Reinforcement Learning

Suppose agent must make policy while learning

First approach:
- Start with arbitrary policy
- Apply Q-Learning
- New policy:
  - In state \(s\), choose action \(a\) that maximizes \(Q(a, s)\)

Problem?

Part II

- So far, we've assumed agent had policy
- Now, suppose agent must learn it

Utility of Exploration

- Too easily stuck in non-optimal space
  - "Exploration versus exploitation tradeoff"

- Solution 1
  - With fixed probability perform a random action
- Solution 2
  - Increase expected value of infrequent states
~Worlds Best Player

• Neural network with 80 hidden units
  Used computed features
• 300,000 games against self

Imitation Learning

• What can you do if you have a teacher?
• People are often ...
  ... good at demonstrating a system
  ... bad at specifying exact rewards / utilities

• Idea: Learn the reward function that best
  "explains" demonstrated behavior
• That is, learn reward such that
  demonstrated behavior is optimal wrt. It
• Also called apprenticeship learning, inverse RL

Data Collection

• Length
• Speed
• Road Type
• Lanes
• Accidents
• Construction
• Congestion
• Time of day
• 25 Taxi Drivers
• Over 100,000 miles

Destination Prediction

Courtesy of B. Ziebart
Summary

• Use reinforcement learning when
  Model of world is unknown and/or rewards are delayed
• Temporal difference learning
  Simple and efficient training rule
• Q-learning eliminates need for explicit T model
• Large state spaces can (sometimes!) be handled
  Function approximation, using linear functions
  or neural nets