Reinforcement Learning

Introduction & Passive Learning

Alan Fern

* Based in part on slides by Daniel Weld

So far ….

- Given an MDP model we know how to find optimal policies (for moderately-sized MDPs)
  - Value Iteration or Policy Iteration
- Given just a simulator of an MDP we know how to select actions
  - Monte-Carlo Planning
- What if we don't have a model or simulator?
  - Like when we were babies . . .
  - Like in many real-world applications
  - All we can do is wander around the world observing what happens, getting rewarded and punished
- Enters reinforcement learning

Reinforcement Learning

- No knowledge of environment
  - Can only act in the world and observe states and reward
- Many factors make RL difficult:
  - Actions have non-deterministic effects
    - Which are initially unknown
  - Rewards / punishments are infrequent
    - Often at the end of long sequences of actions
    - How do we determine what action(s) were really responsible for reward or punishment? (credit assignment)
  - World is large and complex
- Nevertheless learner must decide what actions to take
  - We will assume the world behaves as an MDP

Pure Reinforcement Learning vs. Monte-Carlo Planning

- In pure reinforcement learning:
  - the agent begins with no knowledge
  - wanders around the world observing outcomes
- In Monte-Carlo planning
  - the agent begins with no declarative knowledge of the world
  - has an interface to a world simulator that allows observing the outcome of taking any action in any state

- The simulator gives the agent the ability to “teleport” to any state, at any time, and then apply any action
- A pure RL agent does not have the ability to teleport
  - Can only observe the outcomes that it happens to reach

Pure Reinforcement Learning vs. Monte-Carlo Planning

- MC planning is sometimes called RL with a “strong simulator”
  - I.e. a simulator where we can set the current state to any state at any moment
- Pure RL is sometimes called RL with a “weak simulator”
  - I.e. a simulator where we cannot set the state
- A strong simulator can emulate a weak simulator
  - So pure RL can be used in the MC planning framework
  - But not vice versa

Passive vs. Active learning

- Passive learning
  - The agent has a fixed policy and tries to learn the utilities of states by observing the world go by
  - Analogous to policy evaluation
  - Often serves as a component of active learning algorithms
  - Often inspires active learning algorithms
- Active learning
  - The agent attempts to find an optimal (or at least good) policy by acting in the world
  - Analogous to solving the underlying MDP, but without first being given the MDP model

Model-Based vs. Model-Free RL

- *Model based approach to RL:*
  - learn the MDP model, or an approximation of it
  - use it for policy evaluation or to find the optimal policy

- *Model free approach to RL:*
  - derive the optimal policy without explicitly learning the model
  - useful when model is difficult to represent and/or learn

- We will consider both types of approaches

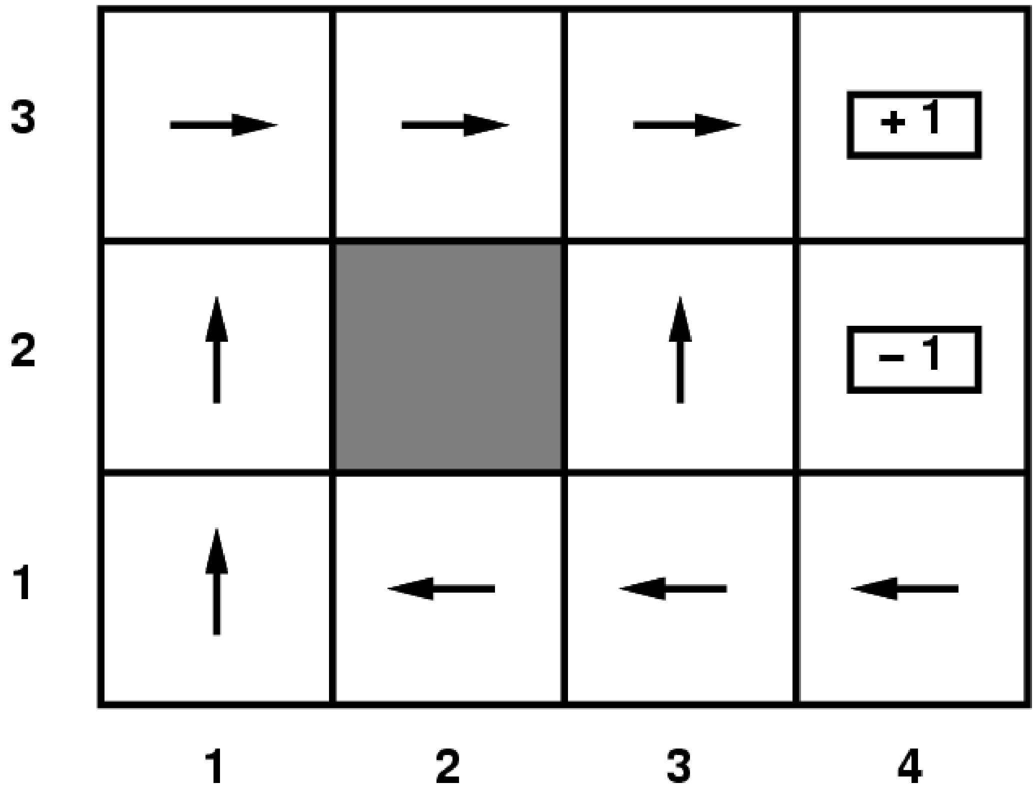
Small vs. Huge MDPs

- We will first cover RL methods for small MDPs
  - MDPs where the number of states and actions is reasonably small
  - These algorithms will inspire more advanced methods

- Later we will cover algorithms for huge MDPs
  - Function Approximation Methods
  - Policy Gradient Methods
  - Least-Squares Policy Iteration

Example: Passive RL

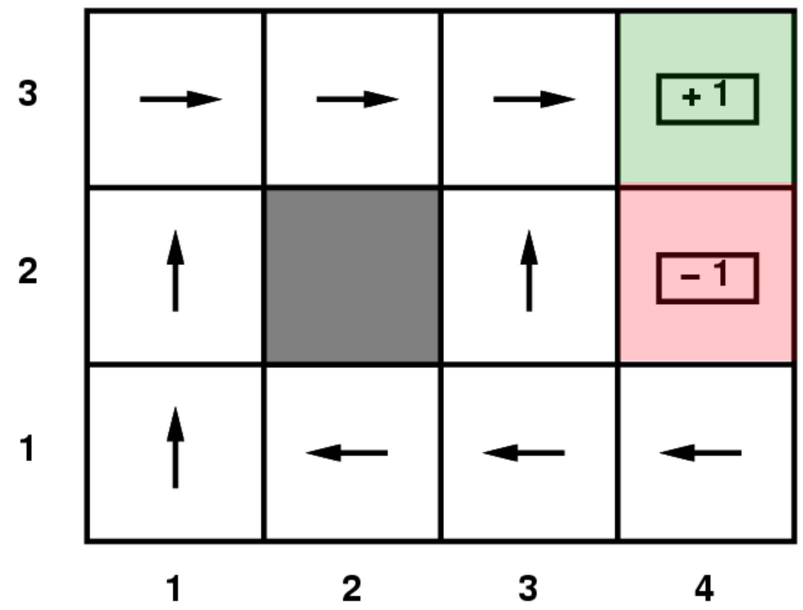
- Suppose given a stationary policy (shown by arrows)
  - Actions can stochastically lead to unintended grid cell
- Want to determine how good it is



Objective: Value Function

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

Passive RL



- Estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!
- Follow the policy for many epochs giving training sequences.

(1,1)→(1,2)→(1,3)→(1,2)→(1,3)→(2,3)→(3,3) →(3,4) +1
(1,1)→(1,2)→(1,3)→(2,3)→(3,3)→(3,2)→(3,3)→(3,4) +1
(1,1)→(2,1)→(3,1)→(3,2)→(4,2) -1

- Assume that after entering +1 or -1 state the agent enters zero reward terminal state
  - So we don't bother showing those transitions

Approach 1: Direct Estimation

- Direct estimation (also called Monte Carlo)
  - Estimate $V^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- ***Reward to go*** of a state s

  the sum of the (discounted) rewards from that state until a terminal state is reached
- Key: use observed ***reward to go*** of the state as the direct evidence of the actual expected utility of that state
- Averaging the reward-to-go samples will converge to true value at state

Direct Estimation

- Converge very slowly to correct utilities values (requires a lot of sequences)

- Doesn't exploit Bellman constraints on policy values

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

  - It is happy to consider value function estimates that violate this property badly.

How can we incorporate the Bellman constraints?

Approach 2: Adaptive Dynamic Programming (ADP)

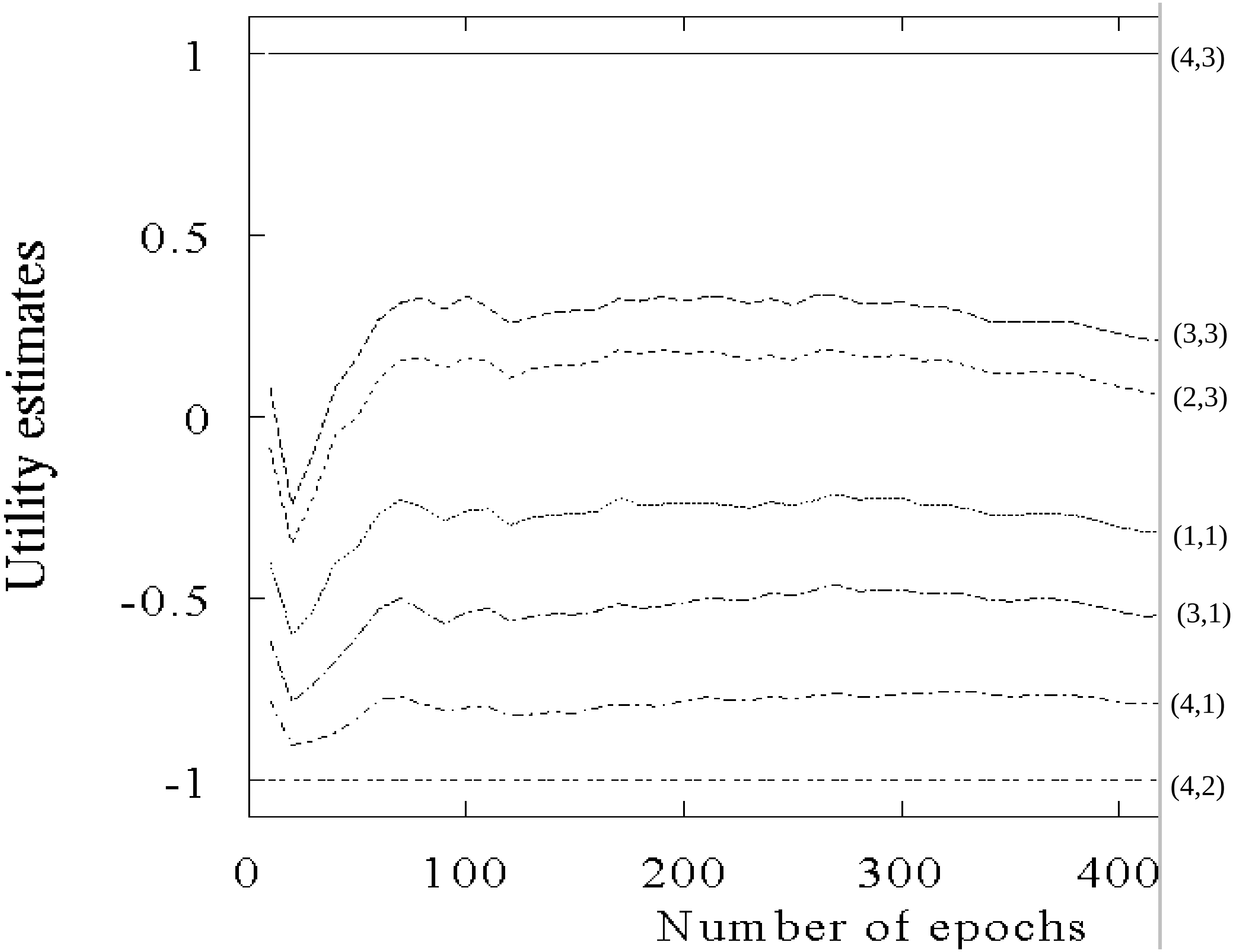
- ADP is a model based approach
  - Follow the policy for awhile
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

learned

- How can we estimate transition model T(s,a,s')?
  - Simply the fraction of times we see s' after taking a in state s.
  - NOTE: Can bound error with Chernoff bounds if we want

ADP learning curves



Approach 3: Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?

- Can we avoid the $O(n^2)$ space requirements for storing the transition model estimate?

- Temporal Difference Learning (model free)
  - Doesn't store an estimate of entire transition function
  - Instead stores estimate of V^{π}, which requires only O(n) space.
  - Does local, cheap updates of utility/value function on a per-action basis

Approach 3: Temporal Difference Learning (TD)

For each transition of π from s to s', update V^{π}(s) as follows:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha(R(s) + \beta V^{\pi}(s') - V^{\pi}(s))$$

updated estimate

learning rate

discount factor

current estimates at s' and s

- Intuitively moves us closer to satisfying Bellman constraint

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

Why?

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i \right)$$

average of n+1 samples

Aside: Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers ($x_1, x_2, x_3, \ldots$)
  - E.g. to estimate the expected value of a random variable from a sequence of samples.

$$\hat{X}_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1} x_i = \frac{1}{n}\sum_{i=1}^{n} x_i + \frac{1}{n+1}\left(x_{n+1} - \frac{1}{n}\sum_{i=1}^{n} x_i \right)$$

$$= \hat{X}_n + \frac{1}{n+1}\left(x_{n+1} - \hat{X}_n \right)$$

average of n+1 samples

learning rate

sample n+1

- Given a new sample x_{n+1}, the new mean is the old estimate (for n samples) plus the weighted difference between the new sample and old estimate

Approach 3: Temporal Difference Learning (TD)

- TD update for transition from s to s':

$$\underbrace{V^\pi(s)}_{\text{updated estimate}} \leftarrow V^\pi(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s) + \beta V^\pi(s')}_{\text{(noisy) sample of value at s based on next state s'}} - V^\pi(s))$$

- So the update is maintaining a "mean" of the (noisy) value samples

- If the learning rate decreases appropriately with the number of samples (e.g. 1/n) then the value estimates will converge to true values! (non-trivial)

$$V^\pi(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^\pi(s')$$

Approach 3: Temporal Difference Learning (TD)

- TD update for transition from s to s':

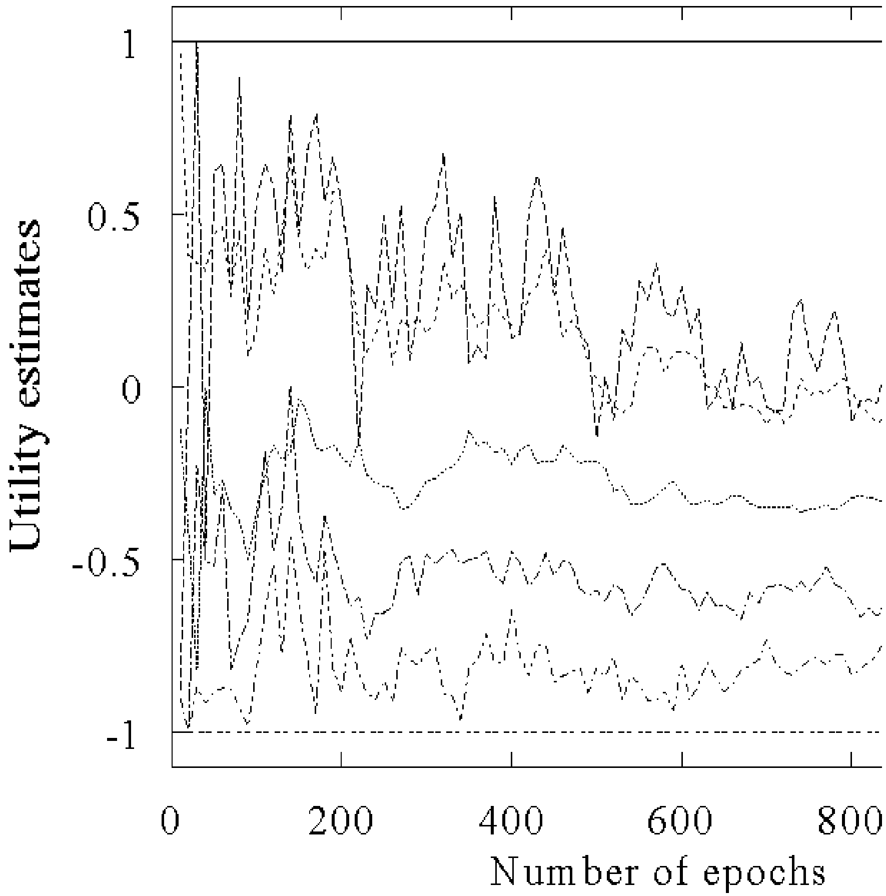
$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s) + \beta V^{\pi}(s')}_{\text{(noisy) sample of utility based on next state}} - V^{\pi}(s))$$

- Intuition about convergence
  - When V satisfies Bellman constraints then **expected** update is 0.

$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s')V^{\pi}(s')$$

  - Can use results from stochastic optimization theory to prove convergence in the limit

The TD learning curve



- **Tradeoff:** requires more training experience (epochs) than ADP but much less computation per epoch
- Choice depends on relative cost of experience vs. computation

Passive RL: Comparisons

- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors as in ADP
  - Convergence in between direct estimation and ADP

Between ADP and TD

- Moving TD toward ADP
  - At each step perform TD updates based on observed transition and "imagined" transitions
  - Imagined transition are generated using estimated model
- The more imagined transitions used, the more like ADP
  - Making estimate more consistent with next state distribution
  - Converges in the limit of infinite imagined transitions to ADP
- Trade-off computational and experience efficiency
  - More imagined transitions require more time per step, but fewer steps of actual experience