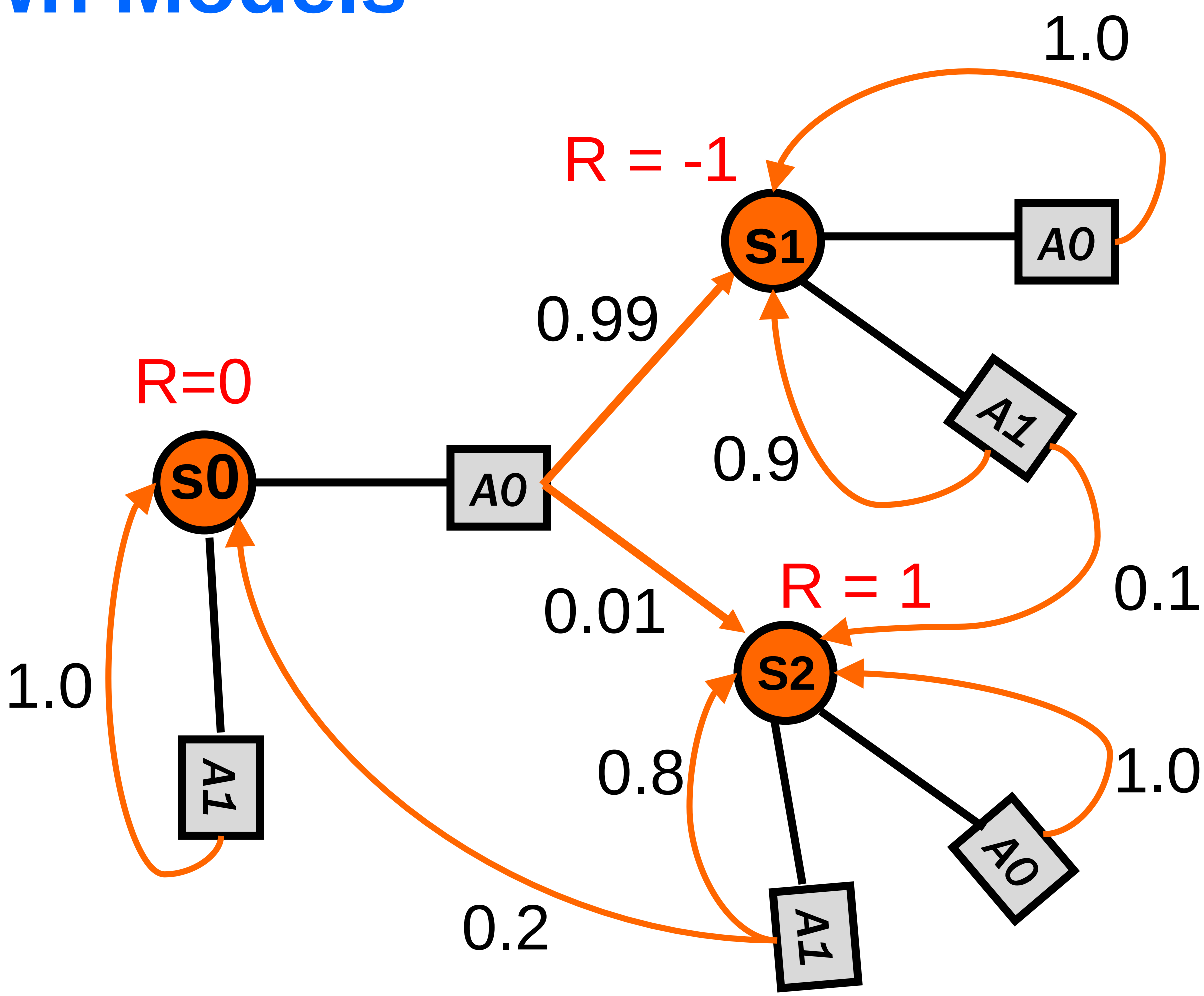
Reinforcement Learning

Introduction & Passive Learning

Alan Fern

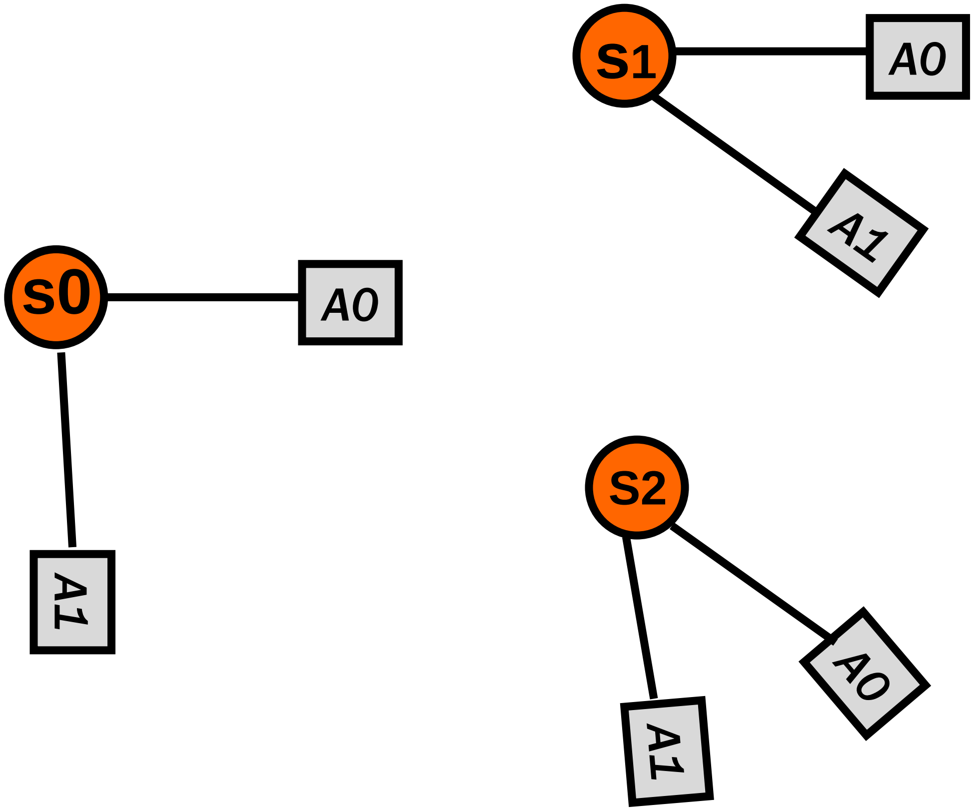
* Based in part on slides by Daniel Weld

Known Models



Given a moderately-sized MDP model, we can use value iteration or value iteration to solve it.

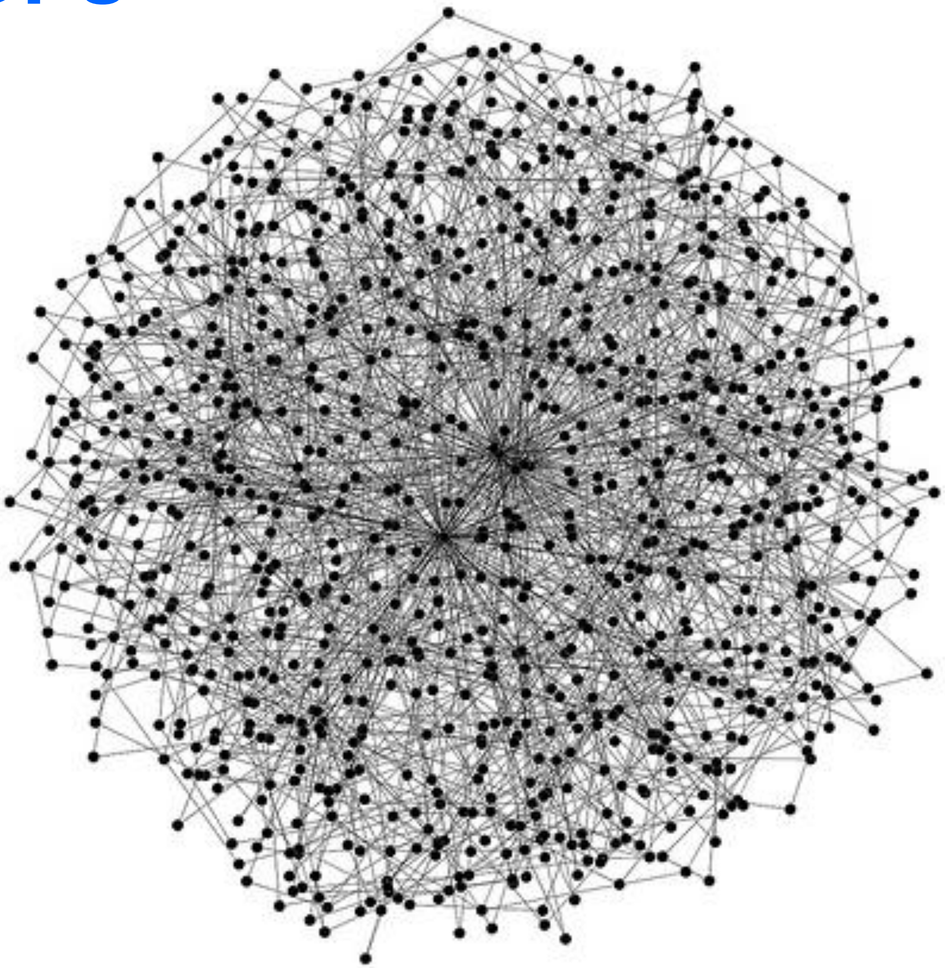
Unknown Models



What if we don't know the reward and transition functions?
(Like in many real-world domains.)

But we can take actions and observe their effects.

Enormous MDPs



What if an MDP is enormous, regardless of whether we know the model or not?

Unknown MDP Model

- In many real-world domains it is difficult to hand-code an MDP model that is sufficiently accurate.

- **Option 1:** Hand-code a parameterized MDP model and then manually collect data to tune the model parameters.
  - E.g. certain probabilities may be unknown, but could be inferred from appropriately collected data

- **Option 2:** Reinforcement learning can do this automatically, or learn a policy directly without explicit model learning.

Enormous Worlds

- We have considered basic model-based planning algorithms

- **Model-based planning**: assumes MDP model is available
  - Methods we learned so far are at least poly-time in the number of states and actions
  - Difficult to apply to large state and action spaces (though this is a rich research area)

- We will consider various methods for overcoming this issue

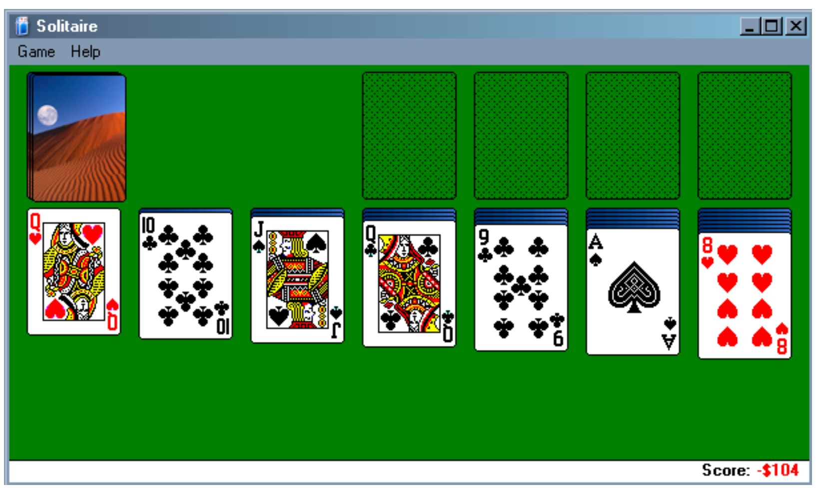
Approaches for Enormous Worlds

- **<u>Planning with compact MDP representations</u>**
  1. Define a language for compactly describing an MDP
     - MDP is exponentially larger than description
     - E.g. via Dynamic Bayesian Networks
  2. Design a planning algorithm that directly works with that language
- Scalability is still an issue
- Can be difficult to encode the problem you care about in a given language
- May study in last part of course

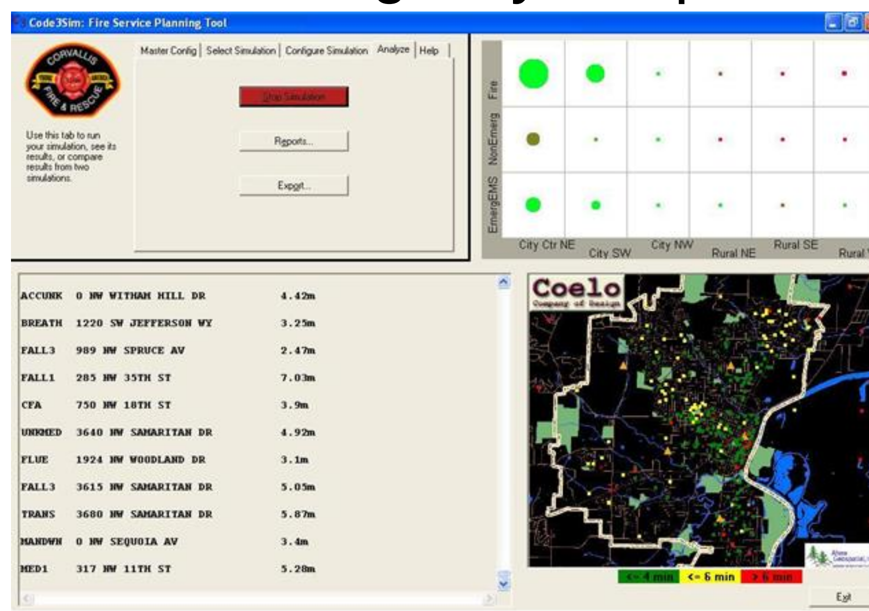
Approaches for Enormous Worlds: Monte-Carlo Planning

- Often a simulator of a planning domain is available or can be learned/estimated from data
  - Will study later in the course

Klondike Solitaire



Fire & Emergency Response



Approaches for Large Worlds

- **Reinforcement learning w/ function approx.**
  1. Have a learning agent directly interact with environment
  2. Learn a compact description of policy or value function

- Often works quite well for large problems
  - Robotics
  - Networking
  - Games (e.g. Atari)
  - ….

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
- Imagine trying to learn to play solitaire or chess without being told the rules or objective

Passive vs. Active learning

- Passive learning (policy evaluation)
  - 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 (policy optimization)
  - 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

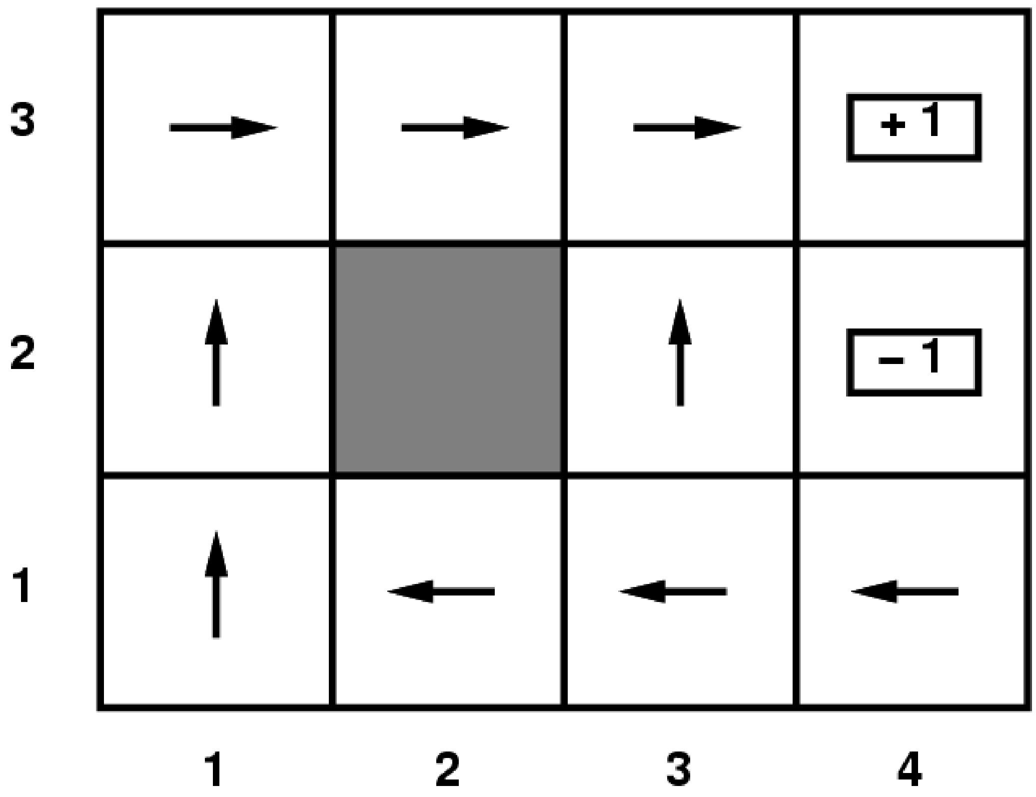
Small vs. Huge MDPs

- 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

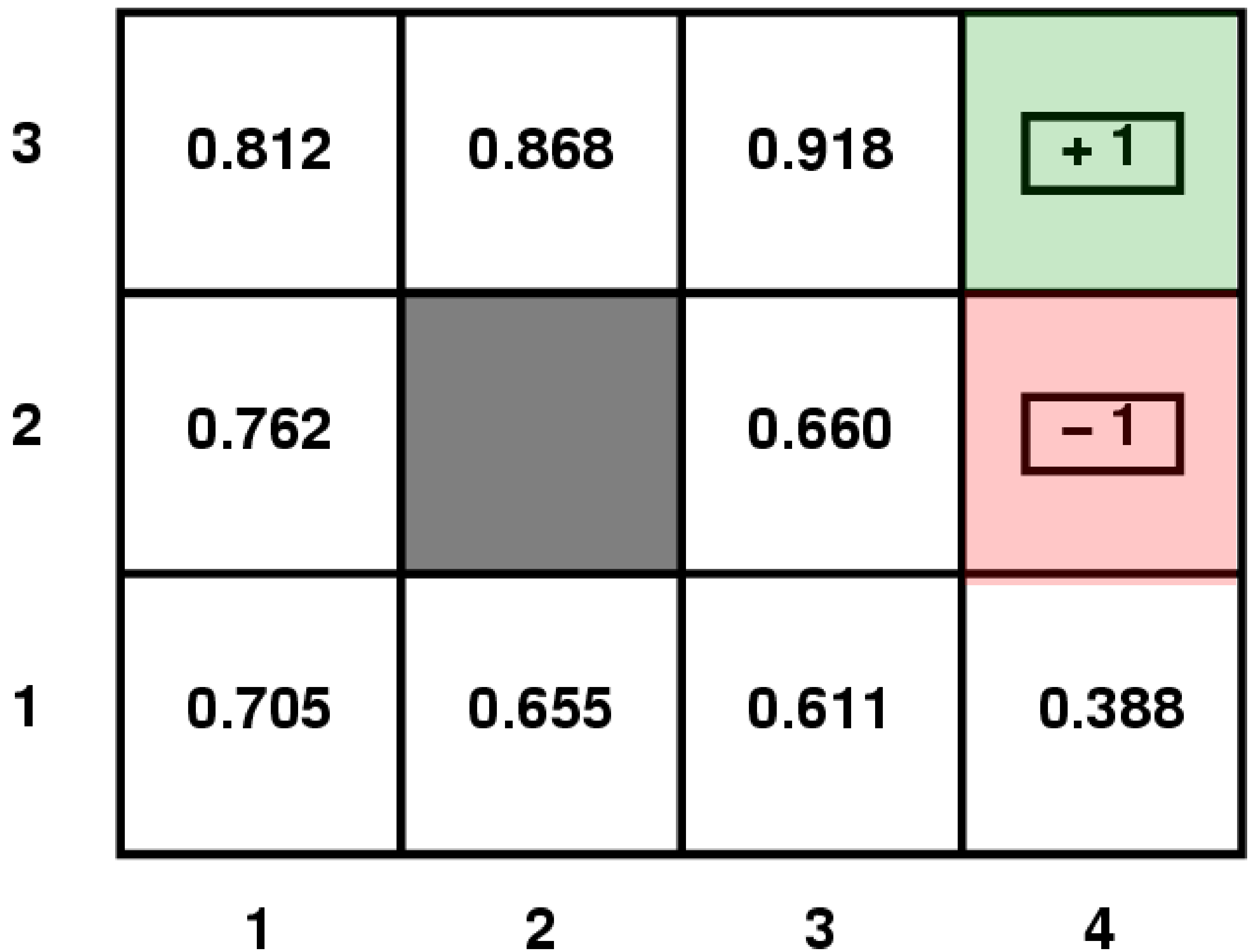
- Later we will cover algorithms for huge MDPs
  - Function Approximation Methods
  - Policy Gradient Methods

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 w/o knowing MDP

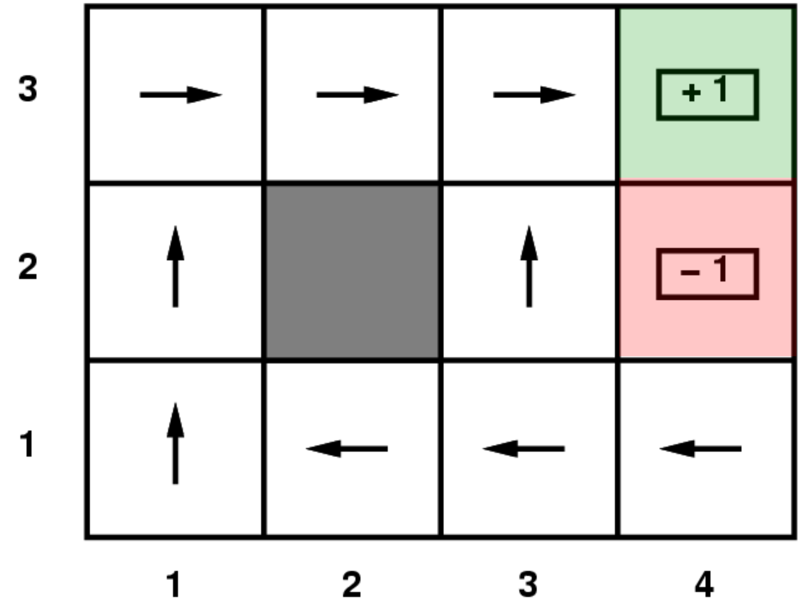


Objective: Value Function



Passive RL

- Estimate $V^{\pi}(s)$
- Not given
  - transition matrix, nor
  - reward function!

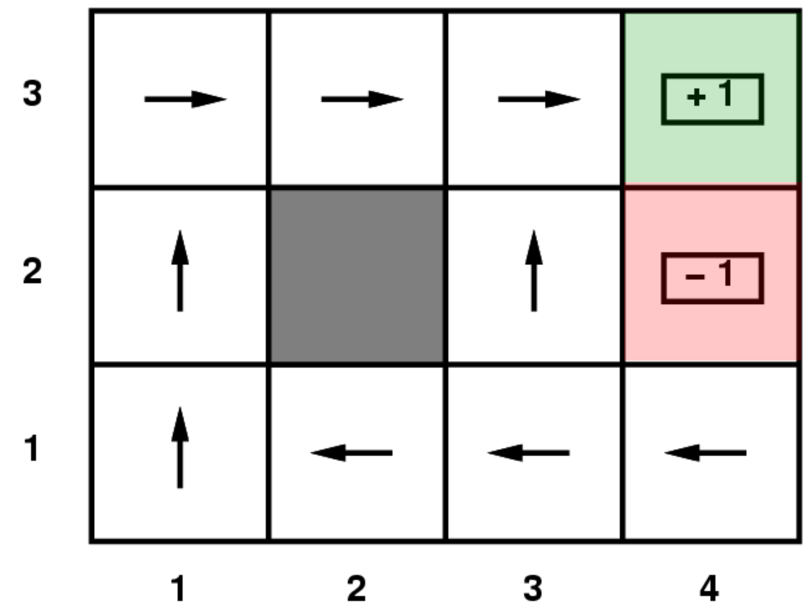


- Follow the policy for many epochs giving training sequences.

(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 episodes end (zero reward everywhere else)

Direct Estimation



(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**

- Estimate $V^{\pi}(s)$ as average future reward observed after state s across trajectories
  - $V^{\pi}\big((1,1)\big) = \frac{1+1-1}{3}$, $V^{\pi}\big((1,2)\big) = \frac{1+1+1}{3}$;; (1,2) is in first trajectory twice

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 slowly to correct values (requires more sequences than perhaps necessary)
  - Is a black box approach. Doesn't recognize that the underlying model is an MDP.

- Doesn't exploit Bellman constraints on policy values

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

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

How can we incorporate the Bellman constraints?

Approach 2: Adaptive Dynamic Programming (ADP)

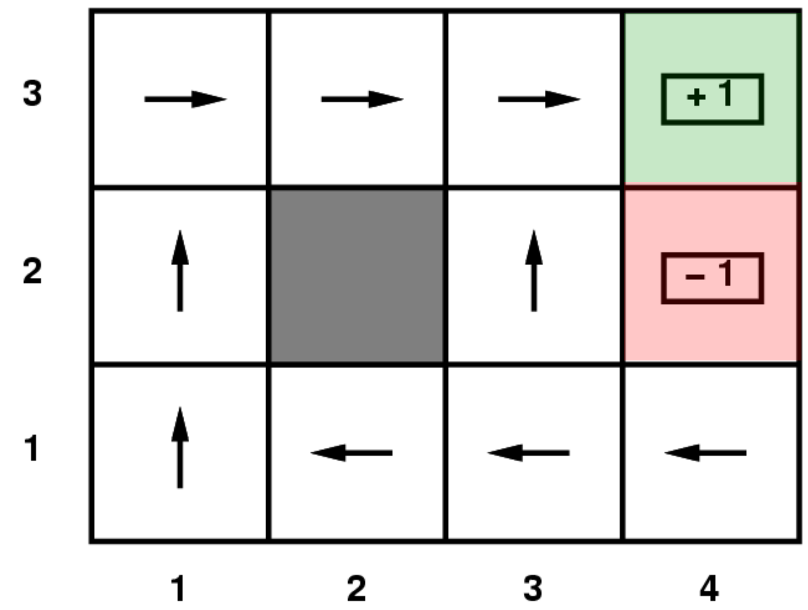
- ADP is a model based approach
  - Follow the policy for awhile
  - Estimate transition model based on observations
  - Learn reward function
  - Use estimated model to compute utility of policy

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

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

Direct Estimation



(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**

- What is estimate of $T\big((1,1), up, (1,2)\big)$?

2/3

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
  - Attempts to respect the Bellman equation

Approach 3: Temporal Difference Learning (TD)

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

$$\underbrace{V^{\pi}(s)}_{\text{updated estimate}} \leftarrow V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(\underbrace{R(s)}_{\text{observed reward}} + \underbrace{\beta}_{\text{discount factor}} V^{\pi}(s') - V^{\pi}(s))$$

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':

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

- The update is maintaining an online average of the (noisy) value samples of $V^{\pi}(s)$
- Why do we call this a noisy sample?

Approach 3: Temporal Difference Learning (TD)

- TD update for transition from s to s':

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

- If each sample $R(s) + \beta V^{\pi}(s')$ was based on the exact value function V^{π}(s'), then the sample average would converge to true value $V^{\pi}(s)$

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

Approach 3: Temporal Difference Learning (TD)

- TD update for transition from s to s':

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

updated estimate

learning rate

- But our samples $R(s) + \beta V^{\pi}(s')$ are based on only our current inexact estimate of $V^{\pi}(s')$.
  - $V^{\pi}(s')$ will be very inaccurate early in learning

Approach 3: Temporal Difference Learning (TD)

- TD update for transition from s to s':

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

updated estimate

learning rate

- Even with these noisy samples, learning will converge to the correct value function V^{π} if α decays appropriately. (non-trivial result)

Approach 3: Temporal Difference Learning (TD)

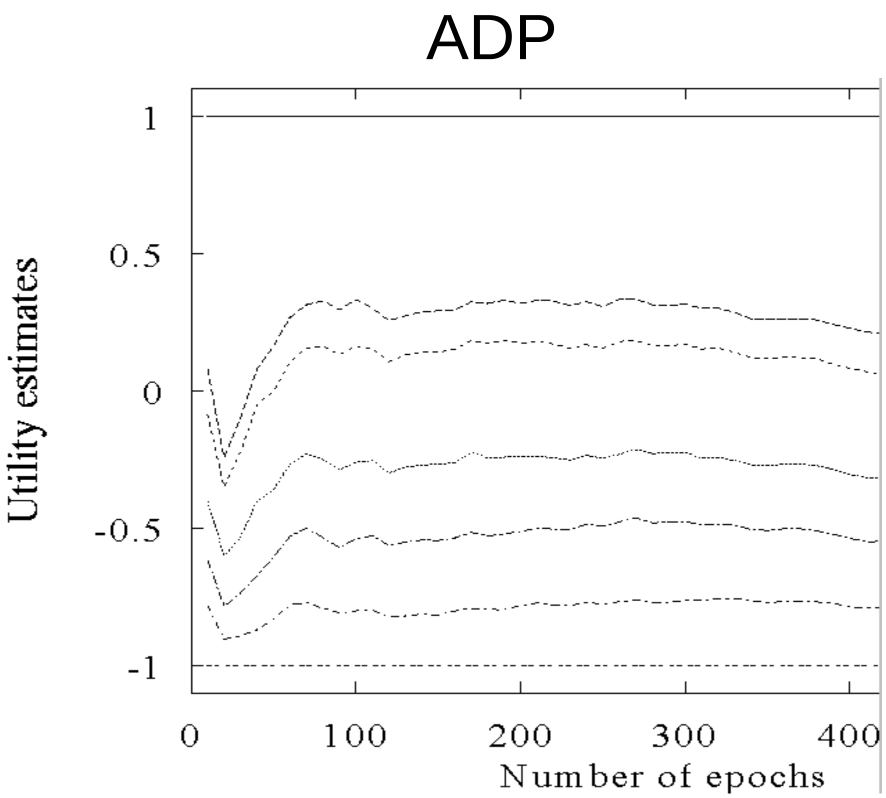
- TD update for transition from s to s':

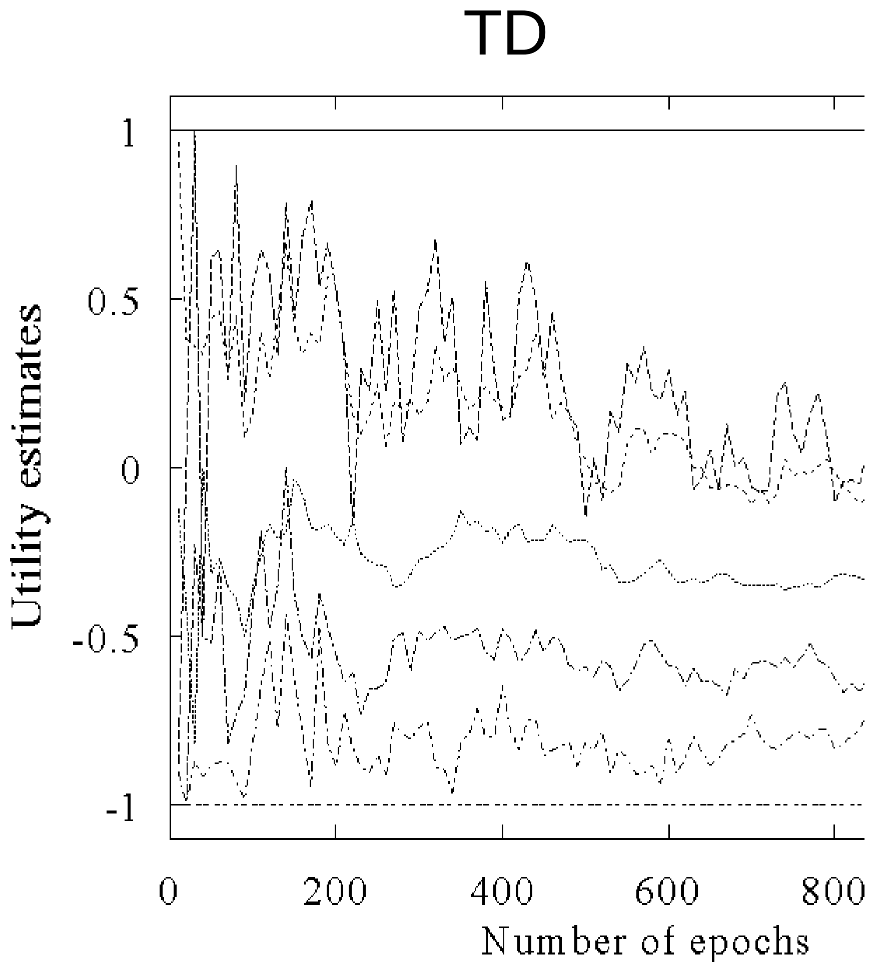
$$\underbrace{V^{\pi}(s)}_{\text{updated estimate}} \leftarrow V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}\big(\underbrace{R(s) + \beta V^{\pi}(s')}_{\text{(noisy) sample of value at s based on next state s'}} - V^{\pi}(s)\big)$$

- Let α_n be the learning rate after n samples
- For convergence we must satisfy:
  - $\sum_{n=1}^{\infty} \alpha_n = \infty$
  - $\sum_{n=1}^{\infty} \alpha_n^2 < \infty$

$\alpha_n = \frac{1}{n}$ is a valid choice

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