CSE 573: Artificial Intelligence

Reinforcement Learning

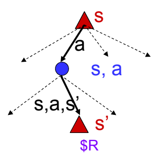
Dan Weld

Many slides adapted from either Alan Fern, Dan Klein, Stuart Russell, Luke Zettlemoyer, or Andrew Moore

Todo

- Add simulations from 473
- Add UCB bound (cut bolzman & constant epsilon
- Add snazzy videos (pendulum, zico kolter…
  - See http://www-inst.eecs.berkeley.edu/~ee128/fa11/videos.html

Markov Decision Processes



Defined as:

- finite state set S
- finite action set A
- Transition distribution $P_T(s' \mid s, a)$
- Bounded reward distribution $P_R(r \mid s, a)$

Policy π

Value & Q functions

Value iteration

Policy Iteration

Q*(a, s)

$$V^*(s) = \max_{a \in Ap(s)} \sum_{s' \in S} Pr(s'|s,a)\left[\mathcal{R}(s,a,s') + \gamma V^*(s')\right]$$

Agent Assets

Transition & Reward Model	Simulator	World
Value Iteration Policy Iteration	Monte Carlo Planning	Reinforcement Learning

Agent Assets

Simulator

Monte Carlo Planning

- •Uniform Monte-Carlo
  - •Single State Case (PAC Bandit)
  - •Policy rollout
  - •Sparse Sampling
- •Adaptive Monte-Carlo
  - •Single State Case (UCB Bandit)
  - •UCT Monte-Carlo Tree Search

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

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
- But learner must decide what actions to take
  - We will assume the world behaves as an MDP

10

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

11

Pure Reinforcement Learning vs. Monte-Carlo Planning

- MC planning aka RL with a "strong simulator"
  - I.e. a simulator which can set the current state
- Pure RL aka RL with a "weak simulator"
  - I.e. a simulator w/o teleport
- A strong simulator can emulate a weak simulator
  - So pure RL can be used in the MC planning framework
  - But not vice versa

12

Applications

- Robotic control
  - helicopter maneuvering, autonomous vehicles
  - Mars rover - path planning, oversubscription planning
  - elevator planning
- Game playing - backgammon, tetris, checkers
- Neuroscience
- Computational Finance, Sequential Auctions
- Assisting elderly in simple tasks
- Spoken dialog management
- Communication Networks – switching, routing, flow control
- War planning, evacuation planning

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

14

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 optimal policy w/o explicitly learning the model
  - useful when model is difficult to represent and/or learn
- We will consider both types of approaches

15

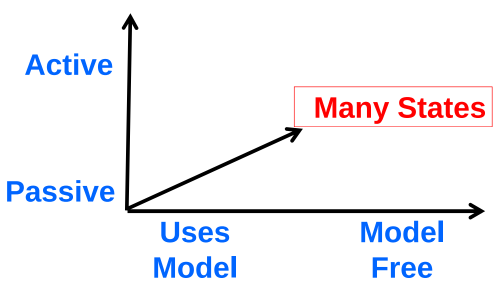
Small vs. Huge MDPs

- First cover RL methods for small MDPs
  - Number of states and actions is reasonably small
    - Eg can represent policy as explicit table
  - 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

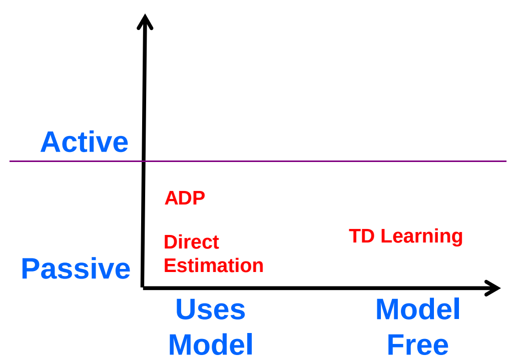
Key Concepts

- Exploration / Exploitation
- GLIE

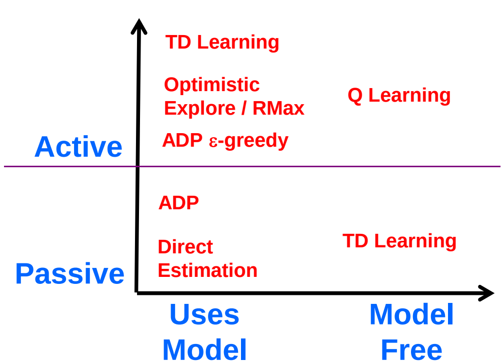
RL Dimensions



RL Dimensions

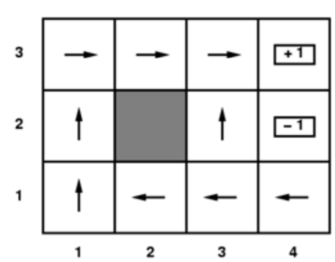


RL Dimensions



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

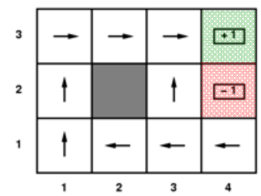


Objective: Value Function

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

22

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

23

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

24

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?

25

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

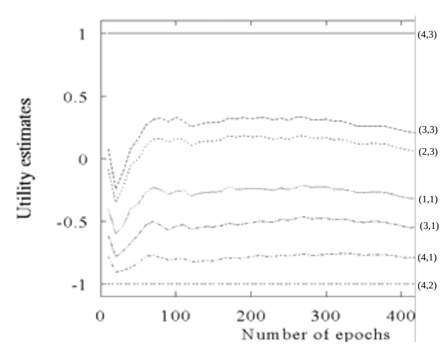
$$V^{\pi}(s) = R(s) + \beta \sum_{s'} T(s,a,s') V^{\pi}(s')$$

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

26

ADP learning curves



27

Approach 3: Temporal Difference Learning (TD)

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning (model free)
  - Do local updates of utility/value function on a per-action basis
  - Don't try to estimate entire transition function!
  - For each transition from s to s', we perform the following update:

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

updated estimate   learning rate   discount factor

- 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

- 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

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':

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

updated estimate   learning rate

- 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':

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

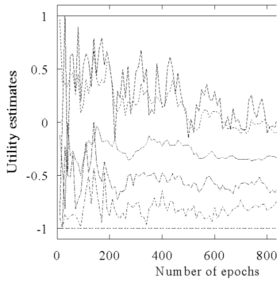
learning rate

- Intuition about convergence
  - When V satisfies Bellman constraints then **<u>expected</u>** 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

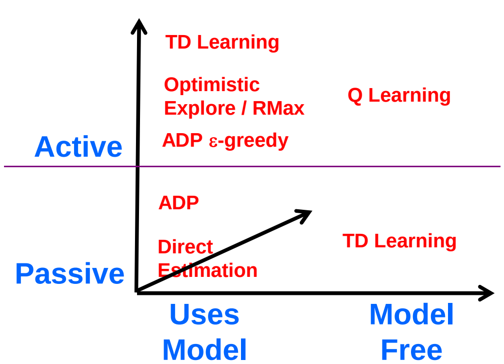
Between ADP and TD

- 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

Break

Unbeknownst to most students of psychology, Pavlov's first experiment was to ring a bell and cause his dog to attack Freud's cat.

RL Dimensions



Active Reinforcement Learning

- So far, we've assumed agent ***has*** a policy
  - We just learned how good it is
- Now, suppose agent must learn a good policy (ideally optimal)
  - While acting in uncertain world

Naïve Model-Based Approach

1. Act Randomly for a (long) time
   - Or systematically explore all possible actions
2. Learn
   - Transition function
   - Reward function
3. Use value iteration, policy iteration, …
4. Follow resulting policy thereafter.

Will this work? Yes (if we do step 1 long enough and there are no "dead-ends")

Any problems? We will act randomly for a long time before exploiting what we know.

40

Revision of Naïve Approach

1. Start with initial (uninformed) model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Execute action suggested by policy in current state
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the greedy policy suggested by current value estimate

Will this work? No. Can get stuck in local minima.

What can be done?

41

Exploration versus Exploitation

- Two reasons to take an action in RL
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- To explore we typically need to take actions that do not seem best according to our current model.
- Managing the trade-off between exploration and exploitation is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

42

ADP-based (model-based) RL

1. Start with initial model
2. Solve for optimal policy given current model (using value or policy iteration)
3. Take action according to an explore/exploit policy (explores more early on and gradually uses policy from 2)
4. Update estimated model based on observed transition
5. Goto 2

This is just ADP but we follow the explore/exploit policy

Will this work? Depends on the explore/exploit policy.

Any ideas?

43

Explore/Exploit Policies

- Greedy action is action maximizing estimated Q-value

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

  - where V is current optimal value function estimate (based on current model), and R, T are current estimates of model
  - Q(s,a) is the expected value of taking action a in state s and then getting the estimated value V(s') of the next state s'
- Want an exploration policy that is greedy in the limit of infinite exploration (GLIE)
  - Guarantees convergence
- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow

44

Explore/Exploit Policies

- GLIE Policy 1
  - On time step t select random action with probability p(t) and greedy action with probability 1-p(t)
  - p(t) = 1/t will lead to convergence, but is slow
- In practice it is common to simply set p(t) to a small constant ε (e.g. ε=0.1 or ε=0.01)
  - Called ε-greedy exploration

45

Explore/Exploit Policies

- GLIE Policy 2: Boltzmann Exploration
  - Select action a with probability,

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

  - T is the temperature. Large T means that each action has about the same probability. Small T leads to more greedy behavior.
  - Typically start with large T and decrease with time

46

The Impact of Temperature

$$\Pr(a \mid s) = \frac{\exp\left(Q(s,a)/T\right)}{\sum_{a' \in A} \exp\left(Q(s,a')/T\right)}$$

- Suppose we have two actions and that Q(s,a1) = 1, Q(s,a2) = 2
- T=10 gives Pr(a1 | s) = 0.48, Pr(a2 | s) = 0.52
  - Almost equal probability, so will explore
- T= 1 gives Pr(a1 | s) = 0.27, Pr(a2 | s) = 0.73
  - Probabilities more skewed, so explore a1 less
- T = 0.25 gives Pr(a1 | s) = 0.02, Pr(a2 | s) = 0.98
  - Almost always exploit a2

47

Alternative Model-Based Approach: Optimistic Exploration

1. Start with initial model
2. Solve for "optimistic policy"
   (uses optimistic variant of value iteration)
   (inflates value of actions leading to unexplored regions)
3. Take greedy action according to optimistic policy
4. Update estimated model
5. Goto 2

Basically act as if all "unexplored" state-action pairs are maximally rewarding.

48

Optimistic Exploration

- Recall that value iteration iteratively performs the following update at all states:

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s') V(s')$$

  - Optimistic variant adjusts update to make actions that lead to unexplored regions look good
- **Optimistic VI:** assigns highest possible value V^{max} to any state-action pair that has not been explored enough
  - Maximum value is when we get maximum reward forever

$$V^{\max} = \sum_{t=0}^{\infty} \beta^t R^{\max} = \frac{R^{\max}}{1-\beta}$$

- What do we mean by "explored enough"?
  - $N(s,a) > N_e$, where N(s,a) is number of times action a has been tried in state s and N_e is a user selected parameter

49

Optimistic Value Iteration

$$V(s) \leftarrow R(s) + \beta \max_a \sum_{s'} T(s,a,s') V(s')$$

Standard VI

- Optimistic value iteration computes an optimistic value function V^+ using following updates

$$V^+(s) \leftarrow R(s) + \beta \max_a \begin{cases} V^{\max}, & N(s,a) < N_e \\ \sum_{s'} T(s,a,s') V^+(s'), & N(s,a) \geq N_e \end{cases}$$

- The agent will behave initially as if there were wonderful rewards scattered all over around– **optimistic** .
- But after actions are tried enough times we will perform standard "non-optimistic" value iteration

50

Optimistic Exploration: Review

1. Start with initial model
2. Solve for optimistic policy using optimistic value iteration
3. Take greedy action according to optimistic policy
4. Update estimated model; Goto 2

Can any guarantees be made for the algorithm?

- If N_e is large enough and all state-action pairs are explored that many times, then the model will be accurate and lead to close to optimal policy
- But, perhaps some state-action pairs will never be explored enough or it will take a very long time to do so
- Optimistic exploration is equivalent to another algorithm, Rmax, which has been proven to efficiently converge

51

Optimistic Exploration

- Rmax ≅ optimistic exploration via optimistic VI
- **PAC Guarantee (Roughly speaking):** *There is a value of N_e (depending on n,k, and Rmax), such that with high probability the Rmax algorithm will select at most a polynomial number of action with value less than ε of optimal)*
- RL can be solved in poly-time in n, k, and Rmax!

55

TD-based Active RL

1. Start with initial value function
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.

5. Goto 2

Just like TD for passive RL, but we follow explore/exploit policy

Given the usual assumptions about learning rate and GLIE, TD will converge to an optimal value function!

56

TD-based Active RL

1. Start with initial value function
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Update estimated model
4. Perform TD update

$$V(s) \leftarrow V(s) + \alpha(R(s) + \beta V(s') - V(s))$$

   V(s) is new estimate of optimal value function at state s.

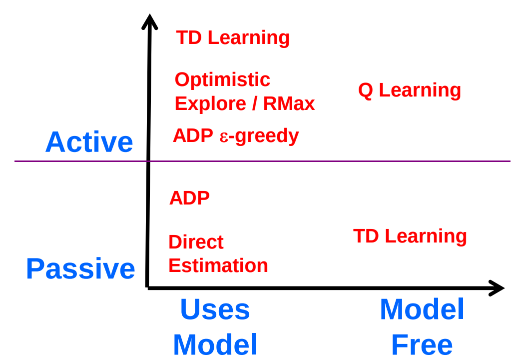
5. Goto 2

Requires an estimated model. Why?

To compute the explore/exploit policy.

57

RL Dimensions



58

TD-Based Active Learning

- Explore/Exploit policy requires computing Q(s,a) for the exploit part of the policy
  - Computing Q(s,a) requires T and R in addition to V
- Thus TD-learning must still maintain an estimated model for action selection
- It is computationally more efficient at each step compared to Rmax (i.e. optimistic exploration)
  - TD-update vs. Value Iteration
  - But model requires much more memory than value function
- Can we get a model-free variant?

59

Q-Learning: Model-Free RL

- Instead of learning the optimal value function V, directly learn the optimal Q function.
  - Recall Q(s,a) is the expected value of taking action a in state s and then following the optimal policy thereafter
- Given the Q function we can act optimally by selecting action greedily according to Q(s,a) ***without a model***
- The optimal Q-function satisfies $V(s) = \max_{a'} Q(s,a')$ which gives:

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

$$= R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$$

How can we learn the Q-function directly?

60

Q-Learning: Model-Free RL

Bellman constraints on optimal Q-function:

$$Q(s,a) = R(s) + \beta \sum_{s'} T(s,a,s') \max_{a'} Q(s,a')$$

- Perform updates after each action just like in TD.
  - After taking action a in state s and reaching state s' do: (note that we directly observe reward R(s))

$$Q(s,a) \leftarrow Q(s,a) + \alpha \underbrace{(R(s) + \beta \max_{a'} Q(s',a') - Q(s,a))}_{\text{(noisy) sample of Q-value based on next state}}$$

Q-Learning

1. Start with initial Q-function (e.g. all zeros)
2. Take action from explore/exploit policy giving new state s' (should converge to greedy policy, i.e. GLIE)
3. Perform TD update
   $$Q(s,a) \leftarrow Q(s,a) + \alpha (R(s) + \beta \max_{a'} Q(s',a') - Q(s,a))$$
   Q(s,a) is current estimate of optimal Q-function.
4. Goto 2

- Does not require model since we learn Q directly!
- Uses explicit |S|x|A| table to represent Q
- Explore/exploit policy directly uses Q-values
  - E.g. use Boltzmann exploration.
  - Book uses exploration function for exploration (Figure 21.8)

Q-Learning: Speedup for Goal-Based Problems

- **Goal-Based Problem:** receive big reward in goal state and then transition to terminal state
- Consider initializing Q(s,a) to zeros and then observing the following sequence of (state, reward, action) triples
  - (s0,0,a0) (s1,0,a1) (s2,10,a2) (terminal,0)
- The sequence of Q-value updates would result in: Q(s0,a0) = 0, Q(s1,a1) =0, Q(s2,a2)=10
- So nothing was learned at s0 and s1
  - Next time this trajectory is observed we will get non-zero for Q(s1,a1) but still Q(s0,a0)=0

Q-Learning: Speedup for Goal-Based Problems

- From the example we see that it can take many learning trials for the final reward to "back propagate" to early state-action pairs
- Two approaches for addressing this problem:
  1. **Trajectory replay**: store each trajectory and do several iterations of Q-updates on each one
  2. **Reverse updates:** store trajectory and do Q-updates in reverse order
- In our example (with learning rate and discount factor equal to 1 for ease of illustration) reverse updates would give
  - Q(s2,a2) = 10, Q(s1,a1) = 10, Q(s0,a0)=10

Active Reinforcement Learning Summary

- Methods
  - ADP
  - Temporal Difference Learning
  - Q-learning
- All converge to optimal policy assuming a GLIE exploration strategy
  - Optimistic exploration with ADP can be shown to converge in polynomial time with high probability
- All methods assume the world is not too dangerous (no cliffs to fall off during exploration)
- So far we have assumed small state spaces

ADP vs. TD vs. Q

- Different opinions….
  - When n is small then doesn't matter much.
- Computation Time
  - ADP-based methods use more computation time per step
- Memory Usage
  - ADP-based methods uses $O(mn^2)$ memory
  - Active TD-learning uses $O(mn^2)$ memory (for model)
  - Q-learning uses O(mn) memory for Q-table
- Learning efficiency (performance per experience)
  - ADP methods reuse experience by reasoning about a learned model (e.g. via value iteration)
  - But … need to learn more parameters (↑ variance)

What about large state spaces?

- One approach is to map the original state space S to a much smaller state space S' via some hashing function.
  - Ideally "similar" states in S are mapped to the same state in S'
- Then do learning over S' instead of S.
  - Note that the world may not look Markovian when viewed through the lens of S', so convergence results may not apply
  - But, still the approach can work if a good enough S' is engineered (requires careful design), e.g.
  - *Empirical Evaluation of a Reinforcement Learning Spoken Dialogue System.* With S. Singh, D. Litman, M. Walker. *Proceedings of the 17th National Conference on Artificial Intelligence*, 2000
- Three other approaches for dealing with large state-spaces
  - **Value function approximation**
  - Policy gradient methods
  - Least Squares Policy Iteration

68