THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 25: Reinforcement Learning

Instructor: Sergei V. Kalinin

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

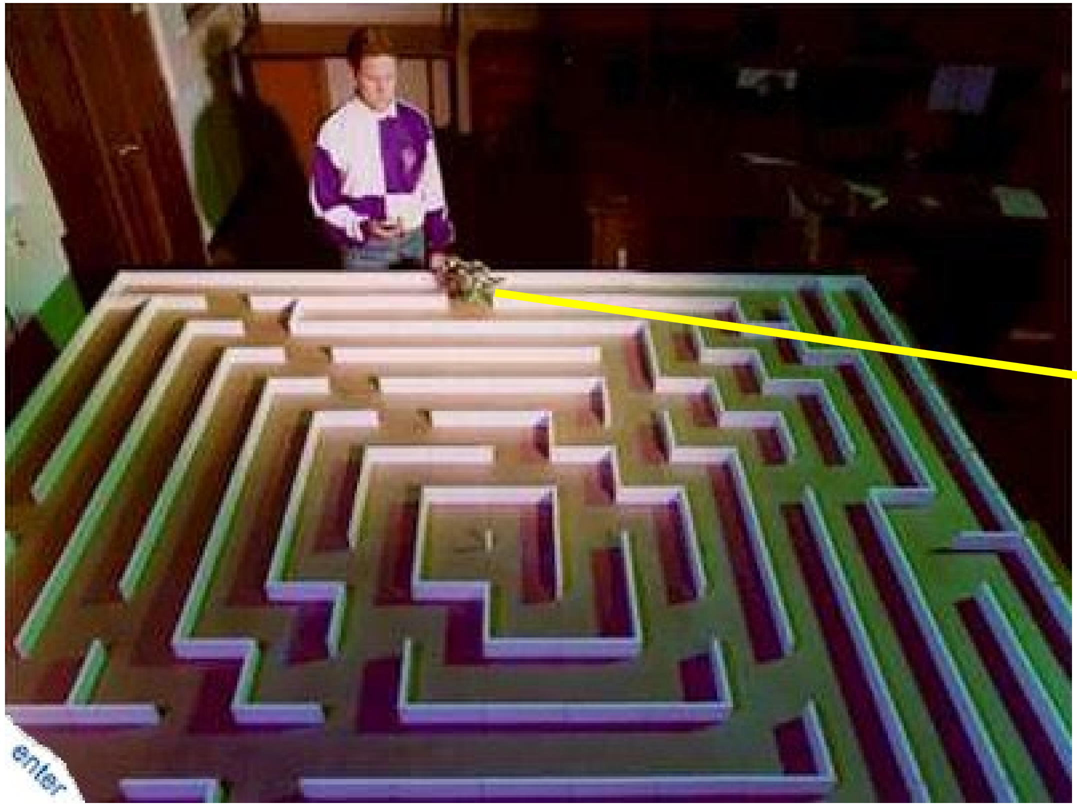
From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

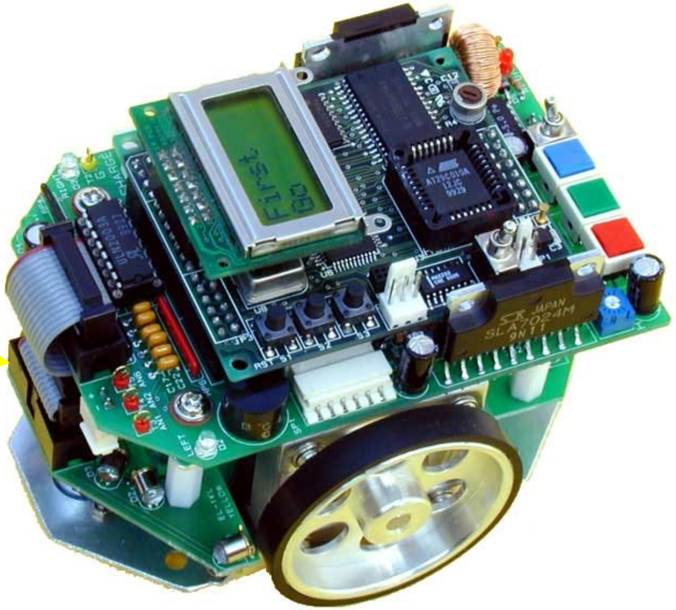
But what if we do not know model?

- Given an MDP model we know how to find optimal policies
  - Value Iteration or Policy Iteration
- But what if we don’t have any form of model
  - In a Maze
  - All we can do is wander around the world observing what happens, getting rewarded and punished
- Enters reinforcement learning

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

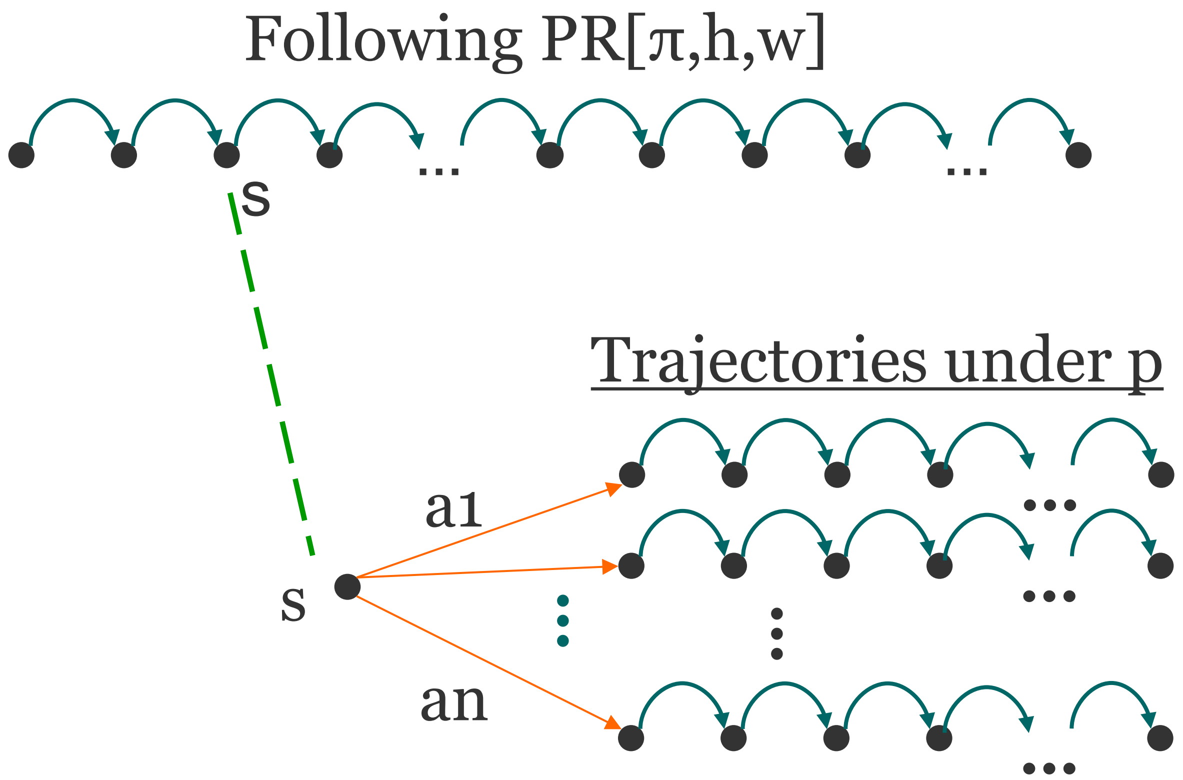
Micromouse





From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Policy roll-out

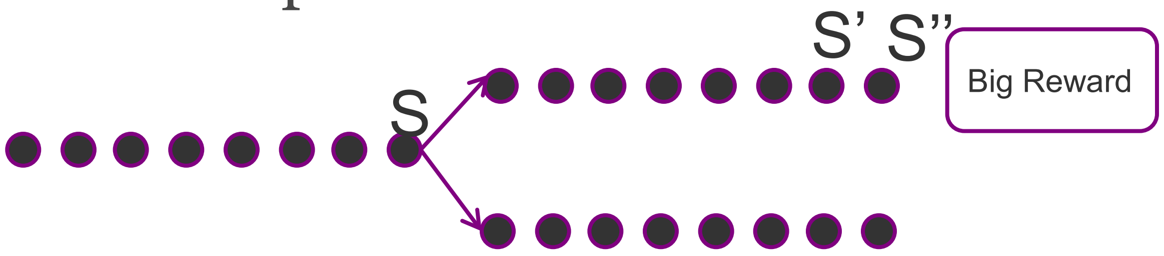


- To compute PR[π,h,w](s) for each action we need to compute w trajectories of length h
- Total of |A|hw calls to the simulator

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Delayed Reward makes it hard to learn

- Delayed Reward Make it hard to learn
- The choice in State S was important but, it seems the action in S' lead to the big reward in S"
- How you deal with this problem?



- How about the performance comparison of VI and PI in this example?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Building RL problem

- How we update the value function or policy?
  – How do we form training data
  – Sequence of (s,a,r)….
- How we explore?
  – Exploit or Exploration

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Categories of RL

- Model-based RL
  – Constructs domain transition model, MDP
- Model-free RL
  – Only concerns policy Q-Learning

- Active Learning (Off-Policy Learning)
  – Q-Learning
- Passive Learning (On-Policy learning)
  – SARSA

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Policy Evaluation

- Remember the formula?

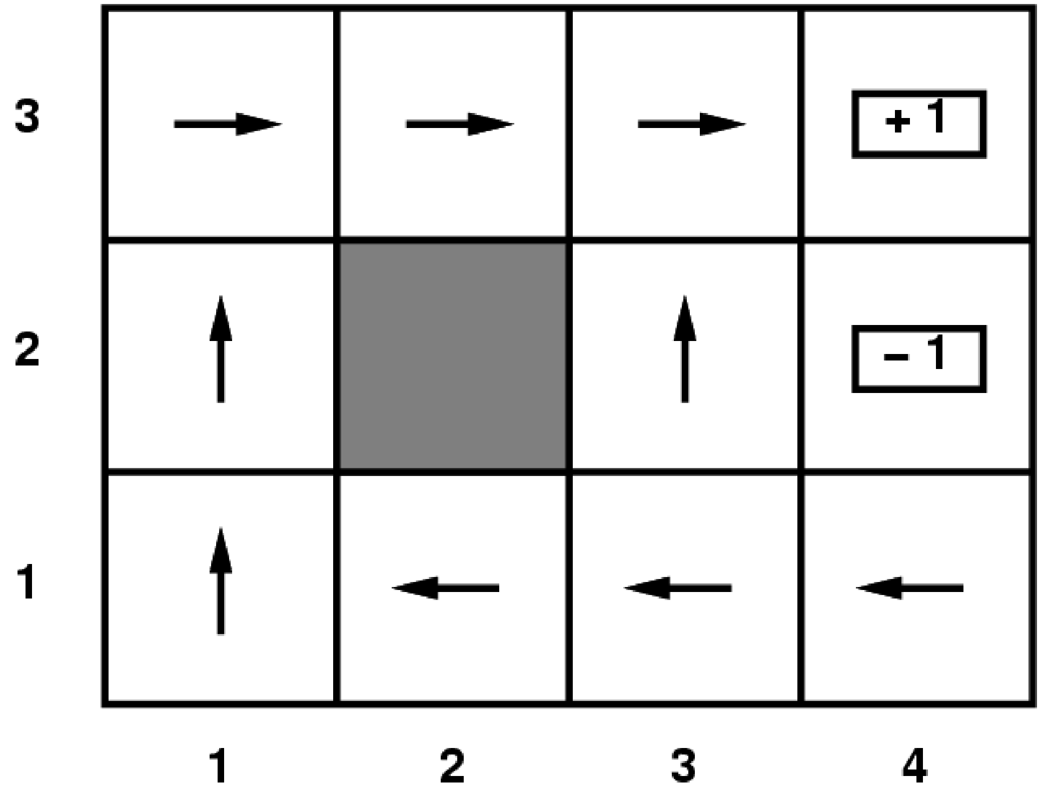
$$V_\pi(s) = R(s) + \beta \sum_{s'} T(s, \pi(s), s') \cdot V_\pi(s')$$

- Can we do that with RL?
  - What is missing?
  - What needs to be done?

- What do we do after policy evaluation?
  - Policy Update

Example of passive RL

- Suppose given policy
- Want to determine how good it is



From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

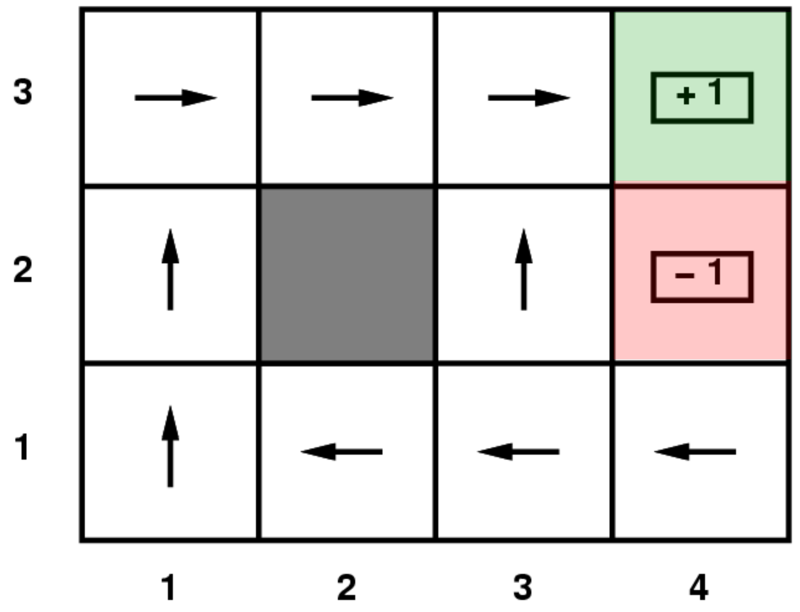
Objective: Value Function

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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Passive RL

- Given policy π,
  – estimate $V^{\pi}(s)$
- Not given
  – transition matrix, nor
  – reward function!
- Simply follow the policy for many epochs
- Epochs: 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

Approach 1: Direct Estimation

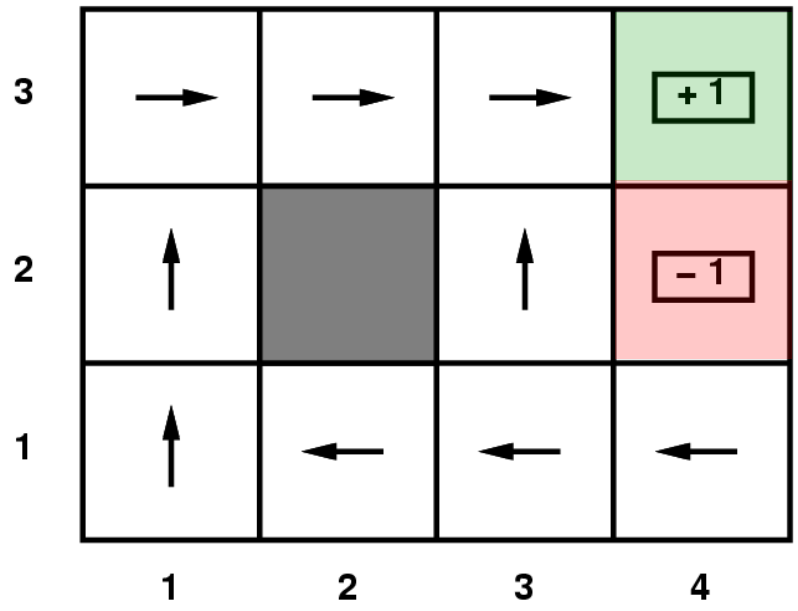
- Direct estimation (model free)
  - Estimate $V^{\pi}(s)$ as average total reward of epochs containing s (calculating from s to end of epoch)
- ***Reward to go*** of a state *s*

  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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Passive RL

- Given policy π,
  – estimate $V^{\pi}(s)$
- Not given
  – transition matrix, nor
  – reward function!
- Simply follow the policy for many epochs
- Epochs: training sequences



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

0.57 0.64 0.72 0.81 0.9

(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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Direct Estimation

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

- Doesn't exploit Bellman constraints on policy values

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

How can we incorporate constraints?

Approach 2: Adaptive Dynamic Programming

- 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 (will see Chernoff bound later in course)

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Approach 3: Temporal Difference (TD) Learning

- Can we avoid the computational expense of full DP policy evaluation?
- Temporal Difference Learning
  - 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:

$$V^{\pi}(s) = V^{\pi}(s) + \underbrace{\alpha}_{\text{learning rate}}(R(s) + \underbrace{\beta}_{\text{discount factor}} V^{\pi}(s') - V^{\pi}(s))$$

- Intuitively moves us closer to satisfying Bellman constraint

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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Online Mean Estimation

- Suppose that we want to incrementally compute the mean of a sequence of numbers
  - 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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Temporal Difference Learning

- TD update for transition from s to s':

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

- So the update is maintaining a "mean" of the (noisy) utility samples
- If the learning rate decreases with the number of samples (e.g. 1/n) then the utility estimates will converge to true values!

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

Temporal Difference Learning

- TD update for transition from s to s':

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

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

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

Comparisons

- **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 similar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors
  - Convergence in between direct estimation and ADP

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Active Reinforcement Learning

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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Naïve 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)

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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Can we do better?

1. Start with initial utility/value function and initial model
2. Take greedy action according to value function|
   (this requires using our estimated model to do "lookahead")
3. Update estimated model
4. Perform step of ADP ; update value function
5. Goto 2

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

Will this work?

No. Gets stuck in local minima.

What can be done?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Exploration vs. 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 exploration and exploitation trade-off is a critical issue in RL
- Basic intuition behind most approaches:
  - Explore more when knowledge is weak
  - Exploit more as we gain knowledge

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

ADP Based RL

1. Start with initial utility/value function
2. Take action according to an explore/exploit policy (explores more early on and gradually becomes greedy)
3. Update estimated model
4. Perform step of ADP
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?

Exploration-Exploitation

- 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 value function estimate, 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

- Solution 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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Exploration-Exploitation

- 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 value function estimate, and R, T are current estimates of model

- Solution 2: Boltzmann Exploration
  – Select action a with probability,

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

  – 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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Alternative: Exploration Functions

1. Start with initial utility/value function
2. Take greedy action
3. Update estimated model
4. Perform step of optimistic ADP
   (uses exploration function in value iteration)
   (inflates value of actions leading to unexplored regions)
5. Goto 2

What do we mean by exploration function in VI?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Exploration Functions

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

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

  – We want to make actions that lead to unexplored regions look good

- Implemented by ***exploration function f(u,n)***:
  – assigning a higher utility estimate to relatively unexplored action state pairs
  – change the updating rule of value function to

$$V^+(s) \leftarrow R(s) + \beta \max_a f\left(\sum_{s'} T(s,a,s')\,V^+(s'), N(a,s)\right)$$

  – *V*+ denote the **optimistic estimate** of the utility
  – N(s,a) = number of times that action a was taken from state s

What properties should f(u,n) have?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Exploration Functions

$$V^{+}(s) \leftarrow R(s) + \beta \max_{a} f\left(\sum_{s'} T(s,a,s')\, V^{+}(s'), N(a,s)\right)$$

- Properties of f(u,n)?
  - If $n > N_e$ u i.e. normal utility
  - Else, R^{+} i.e. max possible value of a state

- 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

- Note that all of these exploration approaches assume that exploration will not lead to unrecoverable disasters (falling off a cliff).

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

TD-based active RL

1. Start with initial utility/value function
2. Take action according to an explore/exploit policy (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!

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

TD-Based Active RL

1. Start with initial utility/value function
2. Take action according to an explore/exploit policy (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.

Goto 2

Requires an estimated model to compute $Q(s,a)$ for greedy policy execution

Can we construct a model-free variant?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Q-Learning: Model Free RL

- Instead of learning the optimal value function V, directly learn the optimal Q function.
  - Recall Q(s,a) is expected value of taking action a in state s and then following the optimal policy thereafter

- The optimal Q-function satisfies $V(s) = \max_{a'} Q(s, a')$ which gives:

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

- Given the Q function we can act optimally by select action greedily according to Q(s,a)

How can we learn the Q-function directly?

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Q-Learning: Model Free RL

Bellman constraints on optimal Q-function:

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

- We can 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))

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

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Q-Learning: Model Free RL

1. Start with initial Q-function (e.g. all zeros)
2. Take action according to an explore/exploit policy (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.

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Direct Policy RL

- Why?
- Again, ironically, policy gradient based approaches were successful in many real applications
- Actually we do this.
  – From 10 Commandments
  – "You Shall Not Murder"
  – "Do not have any other gods before me"
- How we design an policy with parameters?
  – Multinomial distribution of actions in a state
  – Binary classification for each action in a state

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld

Policy Gradient Methods

- $J(\mathbf{w})=E_{\mathbf{w}}[\sum_{t=0..\infty}\gamma^t c_t]$ (failure prob.,makespan, …)
- minimise J by
  - computing gradient $$\nabla J(\mathbf{w}) = \left[\frac{\partial J}{\partial \mathbf{w}_1}, \frac{\partial J}{\partial \mathbf{w}_2}, \ldots, \frac{\partial J}{\partial \mathbf{w}_k}\right]$$
  - stepping the parameters away $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \mathbf{r} J(\mathbf{w})$
- until convergence

- Gradient Estimate Monte Carlo estimate from trace $s_1, a_1, c_1, \ldots, s_T, a_T, c_T$
  - $\mathbf{e}_{t+1} = \mathbf{e}_t + \mathbf{r}_{\mathbf{w}} \log \Pr(a_{t+1}|s_t,\mathbf{w}_t)$
  - $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha\gamma^t c_t \mathbf{e}_{t+1}$
- Successfully used in many applications!

From Sungwook Yoon, Based in part on slides by Alan Fern and Daniel Weld