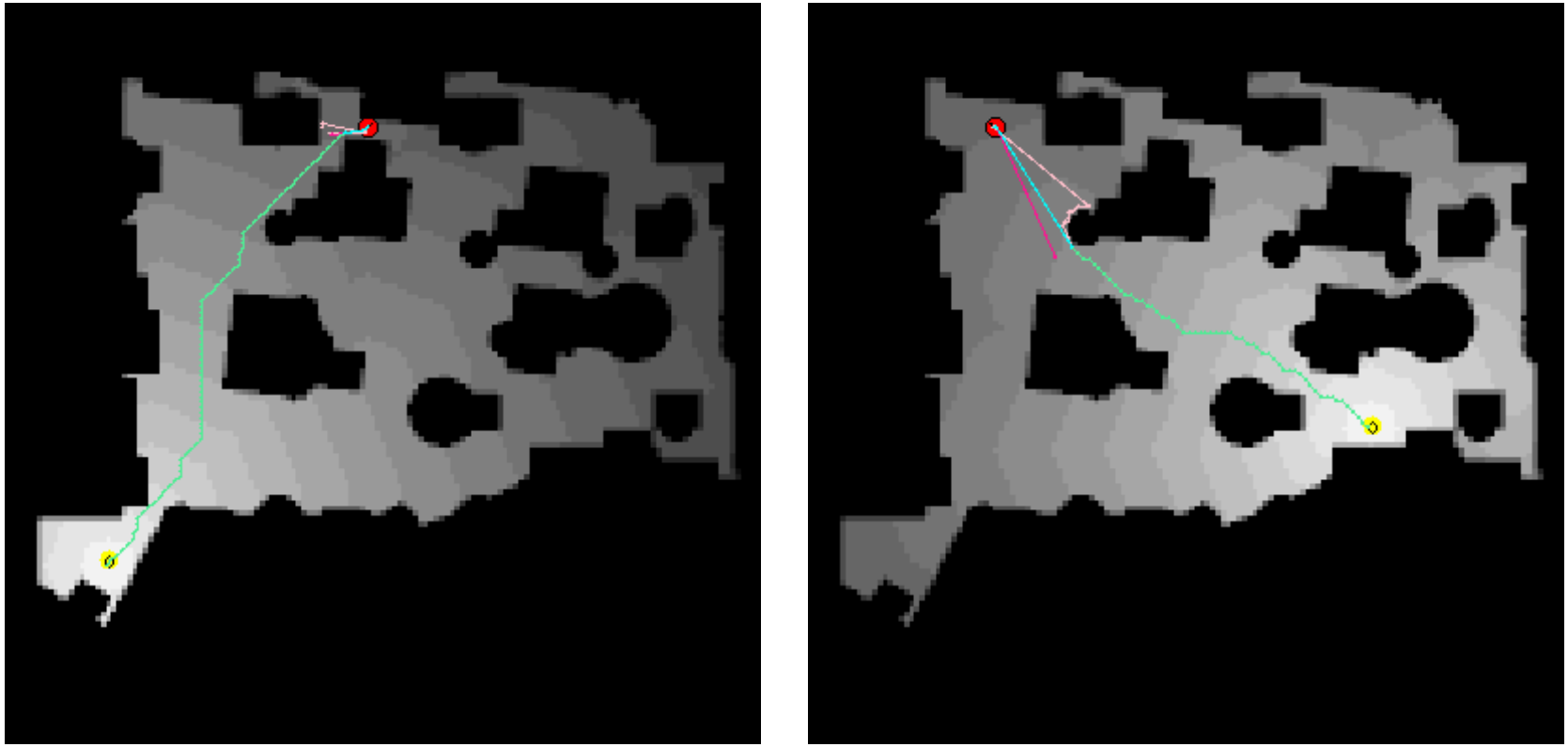
CS 687
Jana Kosecka

Partially Observable MDP's, Reinforcement Learning

Ch. 17, Ch. 21

Value Iteration for Motion Planning



POMDPs

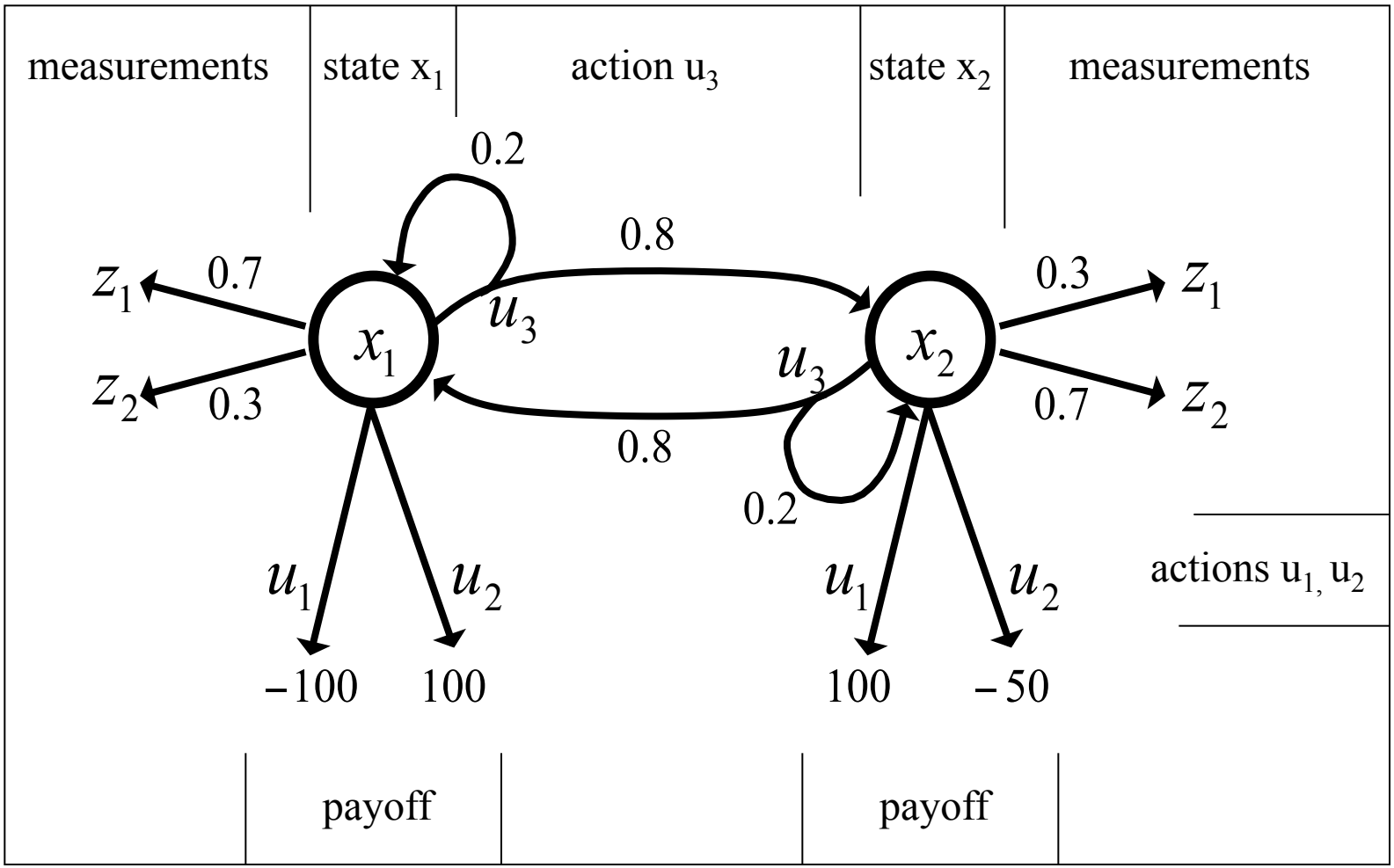
- In POMDPs we apply the very same idea as in MDPs.
- **Since the state is not observable**, the agent has to **make its decisions based on** the belief state which is a **posterior distribution over states**.
- Let b be the belief of the agent about the state under consideration.
- POMDPs compute a **value function over belief space**:

$$V_T(b) \;=\; \gamma \max_u \left[r(b,u) + \int V_{T-1}(b') p(b' \mid u, b)\; db' \right]$$

Problems

- Each belief is a probability distribution, thus, each value in a **POMDP is a function of an entire probability distribution**.
- **This is problematic, since probability distributions are continuous**.
- Additionally, we have to deal with the **huge complexity of belief spaces**.
- For **finite worlds** with finite state, action, and measurement spaces and finite horizons, however, we can **effectively represent the value functions by piecewise linear functions**.

An Illustrative Example



The Parameters of the Example

- The actions u_1 and u_2 are terminal actions.
- The action u_3 is a sensing action that potentially leads to a state transition.
- The horizon is finite and γ=1.

$$
\begin{aligned}
r(x_1, u_1) &= -100 & \qquad r(x_2, u_1) &= +100 \\
r(x_1, u_2) &= +100 & \qquad r(x_2, u_2) &= -50 \\
r(x_1, u_3) &= -1 & \qquad r(x_2, u_3) &= -1
\end{aligned}
$$

$$
\begin{aligned}
p(x_1'|x_1, u_3) &= 0.2 & \qquad p(x_2'|x_1, u_3) &= 0.8 \\
p(x_1'|x_2, u_3) &= 0.8 & \qquad p(z_2'|x_2, u_3) &= 0.2
\end{aligned}
$$

$$
\begin{aligned}
p(z_1|x_1) &= 0.7 & \qquad p(z_2|x_1) &= 0.3 \\
p(z_1|x_2) &= 0.3 & \qquad p(z_2|x_2) &= 0.7
\end{aligned}
$$

Payoff in POMDPs

- In MDPs, the payoff (or return) depended on the state of the system.
- In POMDPs, however, the true state is not exactly known.
- Therefore, we compute the **expected payoff** by **integrating over all states**:

$$
\begin{aligned}
r(b,u) &= E_x[r(x,u)] \\
&= \int r(x,u)p(x)\,dx \\
&= p_1\, r(x_1,u) + p_2\, r(x_2,u)
\end{aligned}
$$

Payoffs in Our Example (1)

- If we are totally certain that we are in state x_1 and execute action u_1, we receive a reward of -100
- If, on the other hand, we definitely know that we are in x_2 and execute u_1, the reward is +100.
- In between it is the linear combination of the extreme values weighted by the probabilities

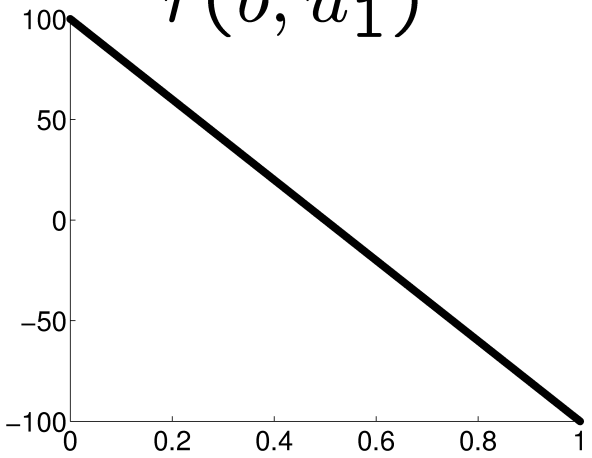
$$
\begin{aligned}
r(b, u_1) &= -100\, p_1 + 100\, p_2 \\
&= -100\, p_1 + 100\, (1 - p_1)
\end{aligned}
$$

$$
r(b, u_2) = 100\, p_1 - 50\, (1 - p_1)
$$

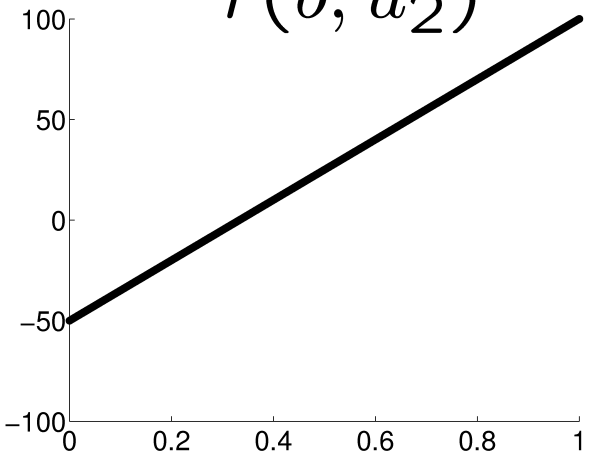
$$
r(b, u_3) = -1
$$

Payoffs in Our Example (2)

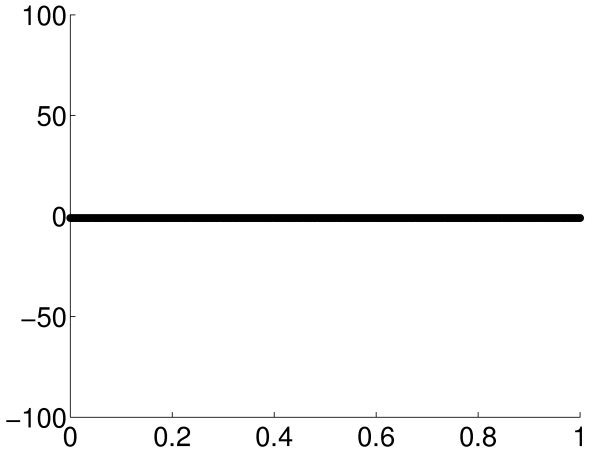
$r(b, u_1)$



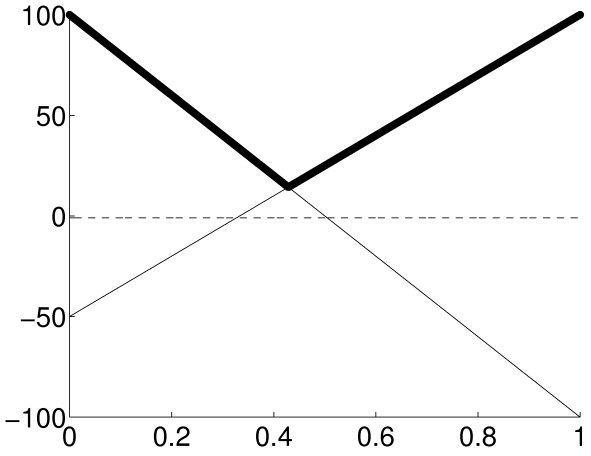
$r(b, u_2)$



$r(b, u_3)$



$V_1(b) = max_u r(b, u)$



The Resulting Policy for T=1

- Given we have a finite POMDP with T=1, we would use $V_1(b)$ to determine the optimal policy.
- In our example, the optimal policy for T=1 is
- the upper thick graph in the diagram.

$$\pi_1(b) = \begin{cases} u_1 & \text{if } p_1 \leq \frac{3}{7} \\ u_2 & \text{if } p_1 > \frac{3}{7} \end{cases}$$

Piecewise Linearity, Convexity

- The resulting value function $V_1(b)$ is the maximum of the three functions at each point

$$
\begin{aligned}
V_1(b) &= \max_u \; r(b,u) \\
&= \max \left\{ \begin{array}{rr} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \\ -1 & \end{array} \right\}
\end{aligned}
$$

- It is piecewise linear and convex.

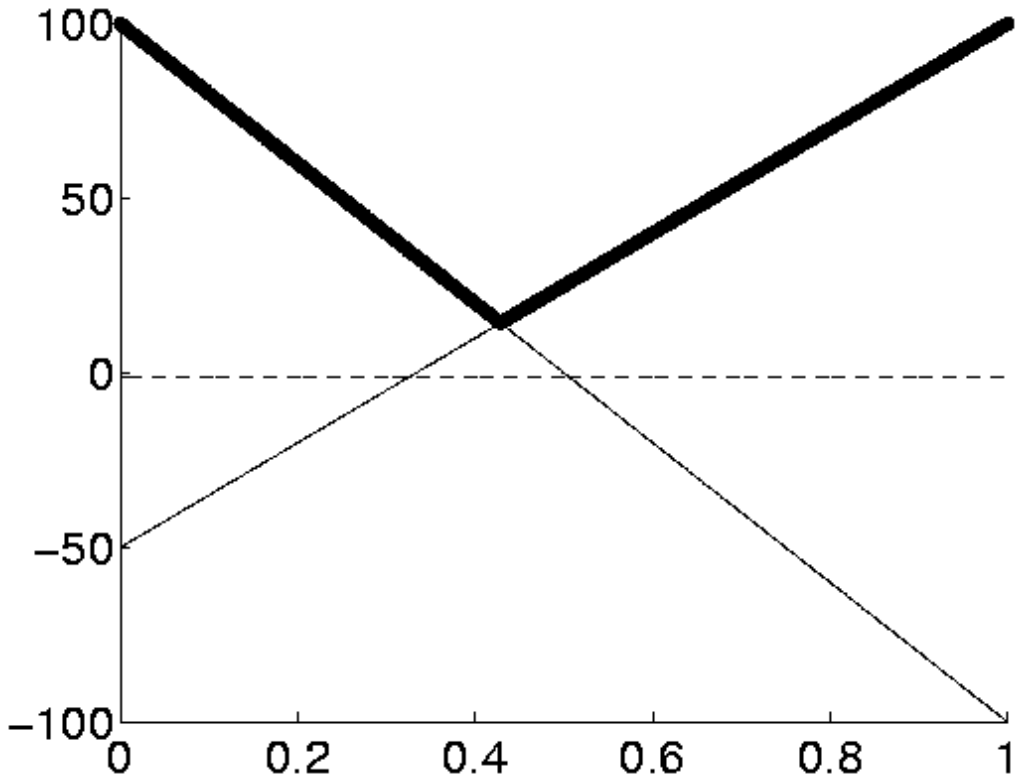
Pruning

- If we carefully consider $V_1(b)$, we see that only the first two components contribute.
- The third component can therefore safely be pruned away from $V_1(b)$.

$$V_1(b) = \max \left\{ \begin{array}{cc} -100\, p_1 & +100\,(1-p_1) \\ 100\, p_1 & -50\,(1-p_1) \end{array} \right\}$$

Increasing the Time Horizon

- Assume the robot can make an observation before deciding on an action.



$V_1(b)$

Reinforcement Learning

- Passive learning – policy is fixed
- learn the utilities of states
- Active learning – learn what to do
- exploration/exploitation
- In many instances the model of the env. and reward are unknown – we just received reward from the env.

Types of environments

- deterministic
- stochastic

Reinforcement Learning

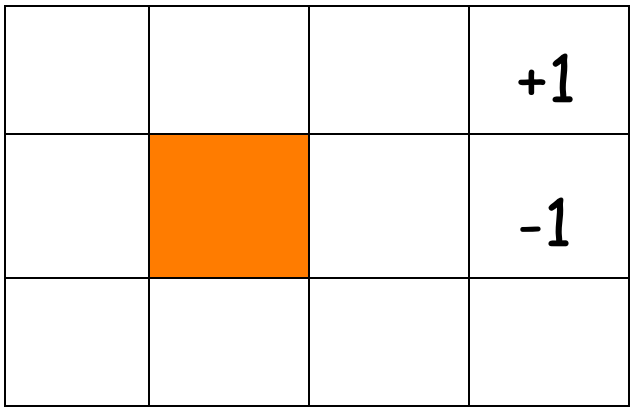
- Examples – baby learning to walk
- Learning how to play a game (chess, backgammon)

- Goal-directed learning from interactions
- Learning what to do
- Goal is encoded as reward signal from the environment
- E.g. learning how to walk – reward proprtional to amout of progress
- Running a maze – reward when successful escape
- We learn to achieve long term goals by interacting with the environment

Reinforcement Learning

- Previously – Markov Decision Processes
- How to make an optimal decisions
- Computation of optimal policies
- Value iteration, Policy iteration
- Model is known
- Transition probabilities, rewards, state is observable

Example: 4x3 world



R(s) = -0.04

T(s,a,s’)

Up = 0.8 up 0.1 left 0.1 right
Left = …
Right = …
Down = …

Reinforcement Learning

- We do not have the model for rewards
- Do not have the transition probabilities
- All we can do is

  1. act 2. perceive state 3. get reward

  Example trials:

(1,1) **-0.04** -> (1,2) **-0.04** -> (1,3) **-0.04** -> …. -> (4,3) **+1**

(1,1) **-0.04** -> (1,2) **-0.04** -> (1,3) **-0.04** -> (2,3) **-0.04** -> (3,3) …. -> (4,3) **+1**

(1,1) **-0.04** -> (2,1) **-0.04** -> (3,1) **-0.04** -> (3,2) **-0.04** -> (4,2) **-1**

Use the information about rewards to learn the utility

Passive Reinforcement Learning

- Learn the utility

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t) | \pi, s_0 = s\right]$$

- Direct utility estimation
- At the end of each sequence calculate the observed reward-to-go for each state and update the utility keep track of the average utility
- over all visits to a particular state
- Does not exploit the information that the utilities of neighboring states are related (via Bellman equation)

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

Adaptive dynamic programming

- How to update the utilities to satisfy the Bellman equation
- We need to know how the states are related
- Need to learn the transition probabilities by keep track of frequencies of reaching *s'* from *s* by executing the action *a*

$$P(s'|s,a) = \frac{\#\text{ of times action a in state s got us to state s'}}{\#\text{ of times we too action a}}$$

- Learn the observed rewards

- Use the Bellman equation to determine the next utility value for each state Learns the utility function faster

Temporal Difference Learning

- Idea: we know that the value function has to satisfy Bellman equation

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

e.g. $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$

Use to observed transitions to satisfy the Bellman equation

- Adjust the value function based on difference between the utilities of successive states

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

- Simpler – instead of doing value determination just update the value

Active reinforcement learning

- Decide what actions to take – no fixed policy
- At each step – follow optimal policy given the current estimate of the utility function
- Greedy agents it mail fail to learn the correct utilities unless it explores also other states
- Choosing always optimal actions can lead overall to suboptimal results
- Fundamental trade-off exploitation (maximize its reward) and exploitation (maximize overall well being)
- Choose random action 1/t times
- Otherwise follow optimal policy – alternatively design some function which will tradeoff greed vs curiosity (taking an action which yields lower utility – but has not been tried often)

Q-learning

- Instead of learning utilities
- learn action value function Q(s,a)

$$Q(a,s) \leftarrow R(s) + \gamma \max_{a'} Q(a',s')$$

- TD Q-learning

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

- TD learning – too expensive to store the value functions
- In large models it is very hard to learn visit every state

Policy search

- Policy - maps states to actions
- We would like to learn directly the policy
- Parameterize the policy by a collection of functions

$$\pi(s) = \max_a Q_\theta(a, s)$$

$$\pi(s) = \text{softmax}_a Q_\theta(a, s)$$