Foundations of Artificial Intelligence

Reinforcement Learning

CS472 – Fall 2007
Thorsten Joachims

Reinforcement Learning

- **Problem**
  - Make sequence of decisions (policy) to get to goal / maximize utility
- **Search Problems so far**
  - Known environment
    - State space
    - Consequences of actions
    - Probability distribution of non-deterministic elements
  - Known utility / cost function
  - First compute the sequence of decisions, then execute (potentially re-compute)
- **Real-World Problems**
  - Environment is unknown a priori and needs to be explored
  - Utility function unknown – only examples are available for some states
    - No feedback on individual actions
    - Learn to act and to assign blame/credit to individual actions
  - Need to quickly react to unforeseen events (have learned what to do)

Reinforcement Learning

- **Issues**
  - Agent knows the full environment a priori vs. unknown environment
  - Agent can be passive (watch) or active (explore)
  - Feedback (i.e. rewards) in terminal states only; or a bit of feedback in any state
  - How to measure and estimate the utility of each action
  - Environment fully observable, or partially observable
  - Have model of environment and effects of action…or not

→ **Reinforcement Learning will address these issues!**

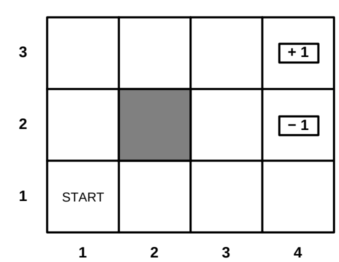
Markov Decision Process

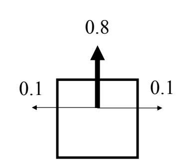
- **Representation of Environment:**
  - finite set of states S
  - set of actions A for each state $s \in S$
- **Process**
  - At each discrete time step, the agent
    - observes state $s_t \in S$ and then
    - chooses action $a_t \in A$.
  - After that, the environment
    - gives agent an immediate reward r_t
    - changes state to s_{t+1} (can be probabilistic)

Markov Decision Process

- **Model:**
  - Initial state: S_0
  - Transition function: T(s,a,s')
    → T(s,a,s') is the probability of moving from state s to s' when executing action a.
  - Reward function: R(s)
    → Real valued reward that the agent receives for entering state s.
- **Assumptions**
  - Markov property: T(s,a,s') and R(s) only depend on current state s, but not on any states visited earlier.
  - Extension: Function R may be non-deterministic as well

Example



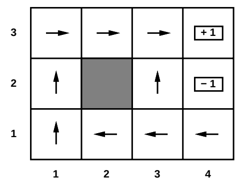


- move into desired direction with prob 80%
- move 90 degrees to left with prob 10%
- move 90 degrees to right with prob 10%

Reward:
- In terminal states reward of +1 / -1 and agent gets "stuck"
- Each other state has a reward of -0.04.

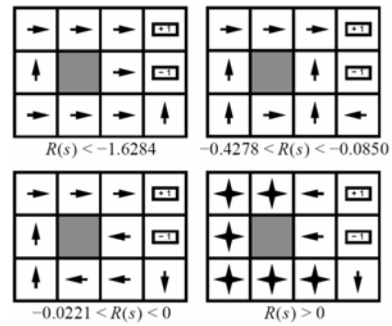
Policy

- **Definition:**
  - A policy π describes which action an agent selects in each state
  - $a=\pi(s)$



- **Utility**
  - For now: $U([s_0,\ldots,s_N]) = \Sigma_i R(s_i)$
  - Let $P([s_0,\ldots,s_N] \mid \pi, s_0)$ be the probability of state sequence $[s_0,\ldots,s_N]$ when following policy π from state s_0
  - Expected utility: $U^\pi(s) = \Sigma\, U([s_0,\ldots,s_N])\, P([s_0,\ldots,s_N] \mid \pi, s_0)$
    → measure of quality of policy π
  - Optimal policy π^*: Policy with maximal $U^\pi(s)$ in each state s

Optimal Policies for Other Rewards



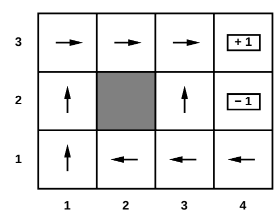
Utility (revisited)

- **Problem:**
  - What happens to utility value when
    - either the state space has no terminal states
    - or the policy never directs the agent to a terminal state

    → Utility becomes infinite
- **Solution**
  - Discount factor $0 < \gamma < 1$
    → closer rewards count more than awards far in the future
  - $U([s_0,\ldots,s_N]) = \Sigma_i\, \gamma^i R(s_i)$
    → finite utility even for infinite state sequences

How to Compute the Utility for a given Policy?

- **Definition:** $U^\pi(s) = \Sigma\, [\, \Sigma_i\, \gamma^i R(s_i)\,]\, P([s_0, s_1,\ldots] \mid \pi, s_0=s)$
- **Recursive computation:**
  - $U^\pi(s) = R(s) + \gamma\, \Sigma_{s'}\, T(s, \pi(s), s')\, U^\pi(s')$



	1	2	3	4
3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388

Here: γ=1.0, R(s)=-0.04

Bellman Update (for fixed π)

Goal: Solve set of n=|S| equations (one for each state)

$U^\pi(s_0) = R(s_0) + \gamma\, \Sigma_{s'}\, T(s_0, \pi(s), s')\, U^\pi(s')$

…

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

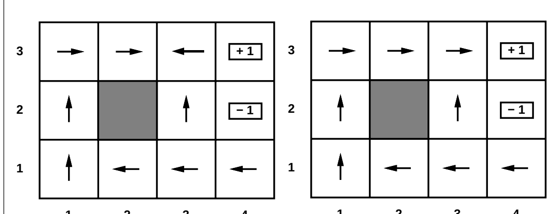
Algorithm [Policy Evaluation]:

- i=0; $U^\pi_0(s)=0$ for all s
- repeat
  - i = i +1
  - for each state s in S do
    - $U^\pi_i(s) = R(s) + \gamma\, \Sigma_{s'}\, T(s, \pi(s), s')\, U^\pi_{i-1}(s')$
  - endfor
- until difference between U^π_i and U^π_{i-1} small enough
- return U^π_i

How to Find the Optimal Policy π^*?

Is policy π optimal? How can we tell?

- If π is not optimal, then there exists some state where
  $\pi(s) \neq \mathrm{argmax}_a\, \Sigma_{s'}\, \mathbf{T(s, a, s')\, U^\pi(s')}$
- How to find the optimal policy π^*?



How to Find the Optimal Policy π^*?

Algorithm [Policy Iteration]:

- repeat
  - U^π = PolicyEvaluation(π,S,T,R)
  - for each state s in S do
    - **If [$\max_a \Sigma_{s'}$ T(s, a, s') U^π(s') > $\Sigma_{s'}$ T(s, π(s), s') U^π(s')] then**
      - » π(s) = $\text{argmax}_a \Sigma_{s'}$ **T(s, a, s') U^π(s')**
  - **endfor**
- until π does not change any more
- return π

Utility ⇔ Policy

Equivalence:

- If we know the optimal utility U(s) of each state, we can derive the optimal policy:

  $\pi^*(s) = \text{argmax}_a \Sigma_{s'}$ **T(s, a, s') U(s')**
- If we know the optimal policy π^*, we can compute the optimal utility of each state:

  PolicyEvaluation algorithm

Bellman Equation:

$U(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s, a, s') U(s')$

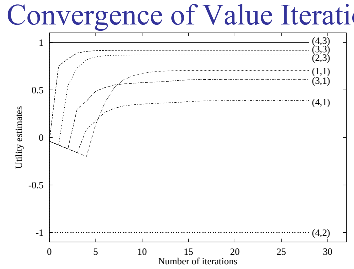
→ Necessary and sufficient condition for optimal U(s).

Value Iteration Algorithm

- **Algorithm [Value Iteration]:**
  - i=0; $U_0(s)$=0 for all s
  - repeat
    - i = i +1
    - for each state s in S do
      - $U_i(s) = R(s) + \gamma \max_a \Sigma_{s'} T(s, a, s') U_{i-1}(s')$
    - endfor
  - until difference between U_i and U_{i-1} small enough
  - return U_i

→ derive optimal policy via $\pi^*(s) = \text{argmax}_a \Sigma_{s'} T(s, a, s') U(s')$

Convergence of Value Iteration



- **Value iteration is guaranteed to converge to optimal U for $0 \leq \gamma < 1$**
- **Faster convergence for smaller γ**

Reinforcement Learning

Assumptions we made so far:

- Known state space S
- Known transition model T(s, a, s')
- Known reward function R(s)

→not realistic for many real agents

Reinforcement Learning:

- Learn optimal policy with a priori unknown environment
- Assume fully observable environment (i.e. agent can tell it's state)
- Agent needs to explore environment (i.e. experimentation)

Passive Reinforcement Learning

Task: Given a policy π, what is the utility function U^π ?

- Similar to Policy Evaluation, but unknown T(s, a, s') and R(s)

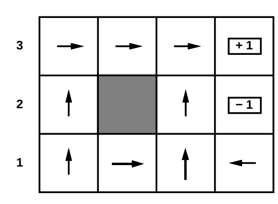
Approach: Agent experiments in the environment

- Trials: execute policy from start state until in terminal state.

$(1,1)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(2,3)_{-0.04}$ → $(3,3)_{-0.04}$ → $(4,3)_{1.0}$

$(1,1)_{-0.04}$ → $(1,2)_{-0.04}$ → $(1,3)_{-0.04}$ → $(2,3)_{-0.04}$ → $(3,3)_{-0.04}$ → $(3,2)_{-0.04}$ → $(3,3)_{-0.04}$ → $(4,3)_{1.0}$

$(1,1)_{-0.04}$ → $(2,1)_{-0.04}$ → $(3,1)_{-0.04}$ → $(3,2)_{-0.04}$ → $(4,2)_{-1.0}$



Direct Utility Estimation

- **Data: Trials of the form**
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
  - $(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1.0}$
- **Idea:**
  - Average reward over all trials for each state independently

    ➔Supervised Learning Problem
- **Why is this less efficient than necessary?**

  ➔ Ignores dependencies between states

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

Adaptive Dynamic Programming (ADP)

- **Idea:**
  - Run trials to learn model of environment (i.e. T and R)
    - Memorize R(s) for all visited states
    - Estimate fraction of times action a from state s leads to s'
  - Use PolicyEvaluation Algorithm on estimated model
- **Problem?**
  - Can be quite costly for large state spaces
  - For example, Backgammon has 10^{50} states

    ➔ Learn and store all transition probabilities and rewards

    ➔ PolicyEvaluation needs to solve linear program with 10^{50} equations and variables.

Temporal Difference (TD) Learning

- **Idea:**
  - Do not learn explicit model of environment!
  - Use update rule that implicitly reflects transition probabilities.
- **Method:**
  - Init $U^\pi(s)$ with R(s) when first visited
  - After each transition, update with

    $U^\pi(s) = U^\pi(s) + \alpha\, [R(s) + \gamma\, U^\pi(s') - U^\pi(s)]$
  - α is learning rate. α should decrease slowly over time, so that estimates stabilize eventually.
- **Properties:**
  - No need to store model
  - Only one update for each action (not full PolicyEvaluation)

Data:
- $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$
- $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{1.0}$

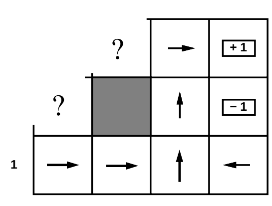
Active Reinforcement Learning

- **Task: In an a priori unknown environment, find the optimal policy.**
  - unknown T(s, a, s') and R(s)
  - Agent must experiment with the environment.
- **Naïve Approach: "Naïve Active PolicyIteration"**
  - Start with some random policy
  - ~~Follow policy~~ to learn model of environment and use ADP to estimate utilities.
  - Update policy using $\pi(s) \leftarrow \text{argmax}_a\, \Sigma_{s'}\, T(s, a, s')\, U^\pi(s')$
- **Problem:**
  - Can converge to sub-optimal policy!
  - By following policy, agent might never learn T and R everywhere.

    ➔ Need for exploration!

Exploration vs. Exploitation

- **Exploration:**
  - Take actions that explore the environment
  - Hope: possibly find areas in the state space of higher reward
  - Problem: possibly take suboptimal steps
- **Exploitation:**
  - Follow current policy
  - Guaranteed to get certain expected reward
- **Approach:**
  - Sometimes take random steps
  - Bonus reward for states that have not been visited often yet



Q-Learning

- **Problem: Agent needs model of environment to select action via**

  $\text{argmax}_a\, \Sigma_{s'}\, T(s, a, s')\, U^\pi(s')$
- **Solution: Learn action utility function Q(a,s), not state utility function U(s). Define Q(a,s) as**

  $U(s) = \max_a Q(a,s)$

  ➔Bellman equation with Q(a,s) instead of U(s)

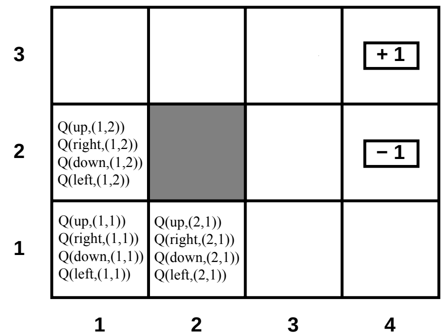
  $Q(a,s) = R(s) + \gamma\, \Sigma_{s'}\, T(s, a, s')\, \max_{a'} Q(a',s')$

  ➔TD-Update with Q(a,s) instead of U(s)

  $Q(a,s) \leftarrow Q(a,s) + \alpha\, [R(s) + \gamma \max_{a'} Q(a',s') - Q(a,s)]$
- **Result: With Q-function, agent can select action without model of environment**

  $\text{argmax}_a\, Q(a,s)$

Q-Learning Illustration



Function Approximation

- **Problem:**
  - Storing Q or U,T,R for each state in a table is too expensive, if number of states is large
  - Does not exploit “similarity” of states (i.e. agent has to learn separate behavior for each state, even if states are similar)
- **Solution:**
  - Approximate function using parametric representation
  - For example: $U(s) = \vec{w} \cdot \Phi(s)$
    - $\Phi(s)$ is feature vector describing the state
      - “Material values” of board
      - Is the queen threatened?
      - …