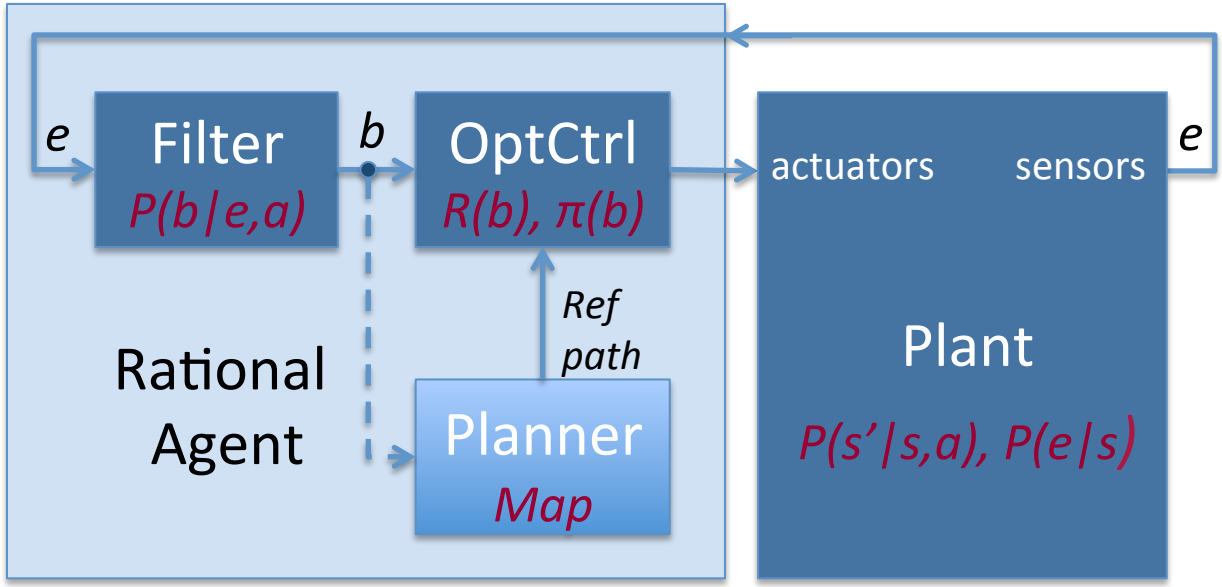
Reinforcement Learning
Chapter 21



Learning of Probabilistic Models

- The Baum Welch Algorithm
  - Feedback: Teacher provides examples (a trace)
  - Supervised learning: This kind of learning technique
- What if such labeled examples are not available?
  - Feedback: Teacher provides rewards (win/loss)
  - Reward or reinforcement: This kind of feedback
  - Reinforcement learning: This kind of learning technique
- Reinforcement examples
  - Chess: Reward only at the end (win or loss)
  - Ping-pong: Reward after every point win or loss

RL Framework

- Reward is part of the input percept
  - Agent: Hardwired to recognize this part as a reward
  - Animals: Recognize pain & hunger as negative R
  - Animals: Recognize pleasure & food as positive R
- Mathematical model
  - MDP: Markov Chain + Inputs + Rewards
  - RL: Use observed rewards to learn optimal policy
  - No prior knowledge: Of the MDP
- RL example
  - Chess: After N moves, you are told: you loose/win
  - Ping-pong: Ball goes out, you are told: loose/win point

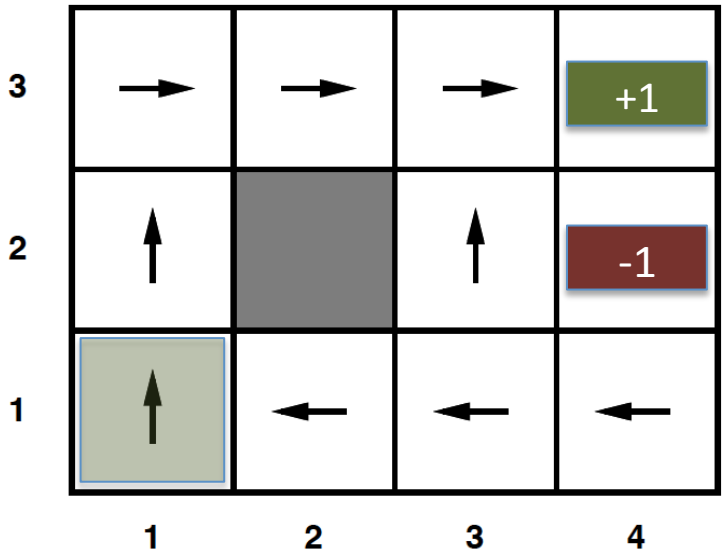
Outline

- Utility-based agents: Learn utility function
- Q-Learning agents: Learn action-utility (Q) function
- Reflex agents: Learn policy

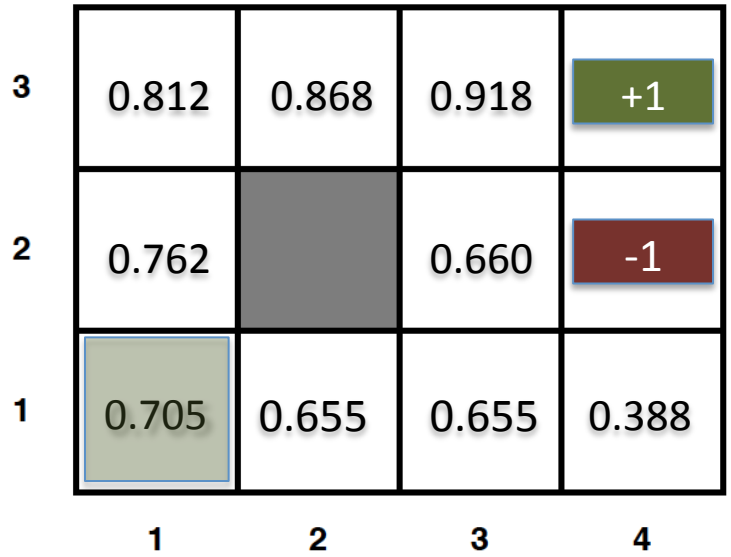
- Passive learning: Policy fixed. Learn utility (model)
- Active learning: Also learn policy (use exploration)

Passive Reinforcement Learning

Given: Policy π (opt for R=-0.04)



Goal: Learn utility U^{π}



Transition model $P(s' \mid s,a)$ Not known

Reward function $R(s)$ Not known

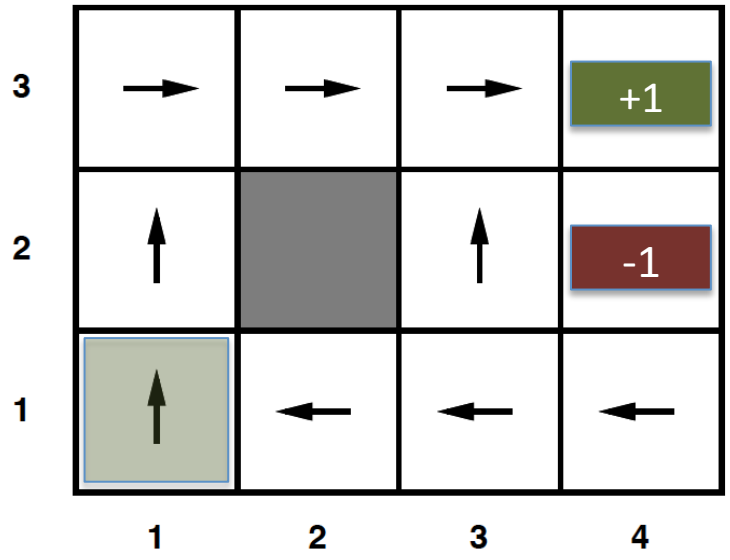
States $(i,j) \in \{1..4\} \times \{1..4\} \setminus (2,2)$ Fully observable

Observations $(i,j)_R$ Reward-indexed states

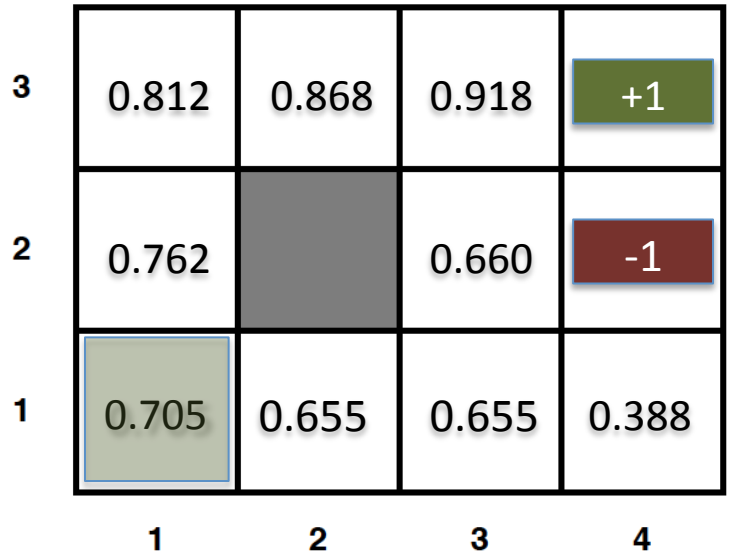
Actions $a \in \{\text{Up},\text{Down},\text{Left},\text{Right}\}$ Known, used by π

Passive Reinforcement Learning

Given: Policy π (opt for R=-0.04)



Goal: Learn utility U^{π}



Trials: executed in the environment using policy π

Trial: Sequence of transitions from (1,1) until terminal state

Percepts: supply both current state and associated reward

$$(1,1)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (2,3)_{-.04} \mapsto (3,3)_{-.04} \mapsto (4,3)_{+1}$$

$$(1,1)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (2,3)_{-.04} \mapsto (3,3)_{-.04} \mapsto (3,2)_{-.04} \mapsto (4,2)_{-1}$$

Objective: Compute $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$ with RV S_t and $S_0 = s$

Direct Utility Estimation

U(s): expected total reward from that state onward (reward-to-go)

One trial: Provides a sample U(s) for each state s visited

$$(1,1)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (2,3)_{-.04} \mapsto (3,3)_{-.04} \mapsto (4,3)_{+1}$$

(1,1) one sample: Of total reward 0.72

(1,2) two samples: Of total rewards 0.76 and 0.84

After each trial: Update U(s) by keeping a running average for $\forall s$

In the limit: Sample average will converge (very slow) to $U^{\pi}(s)$ $\forall s$

Miss: States are not independent but related by Bellman equations

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s))\, U^{\pi}(s')$$

Ignoring Bellman: Misses opportunities for learning

Search space for U: Too large, it includes Us violating Bellman

Adaptive Dynamic Programming (ADP)

ADP: Applies Bellman by learning $P(s'|s,a)$ and using observed $R(s)$

Given π: Bellman equations are linear (no maximization involved)

Model learning: Easy since the environment is fully observable

- Input: (State,action) pair. Output: resulting state
- Simplest representation: Table of probabilities
- Keep track of: How often each a occurs and estimate $P(s'|s,a)$

Example of model learning:

- Right: 3 times in (1,3); 2 times to (2,3). $P((2,3)|(1,3),\text{Right}) = 2/3$
- Model estimation: Maximum-likelyhood technique

Problem: Acts as if the model was correct

Passive ADP Algorithm

function Passive-ADP-Agent (Percept (s', r')) returns Action

persistent Policy π // a fixed policy

persistent MDP M // an MDP $M = (P,R,\gamma)$

persistent Table $U = \varnothing$ // a table of utilities

persistent Table $N_{sa} = \varnothing$ // (state,action) frequencies table

persistent Table $N_{s'|sa} = \varnothing$ // s' frequency for (s,a)

persistent State $s = \text{Null}$, Action $a = \text{Null}$ // previous s and a

if ($s' \notin \text{dom}(U)$) $U[s'] = r'$; $R[s'] = r'$

if ($s \neq \text{Null}$) {

$N_{sa}[s,a]$++; $N_{s'|sa}[s',s,a]$++

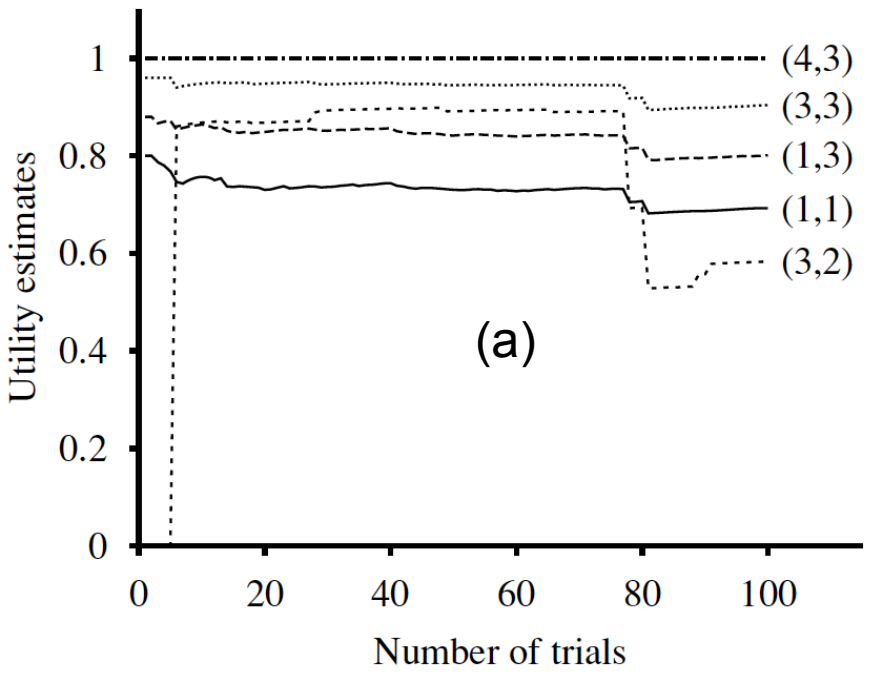
foreach (t st. $N_{s'|sa}[t,s,a] \neq 0$) $P(t \mid s,a) = N_{s'|sa}[t,s,a] / N_{sa}[s,a]$ }

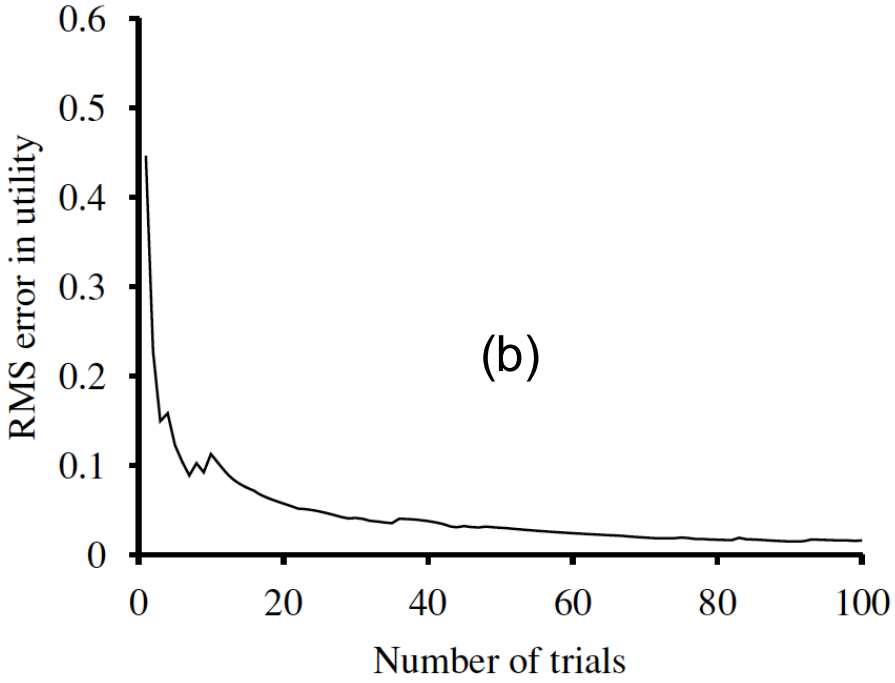
$U = \text{Policy-Evaluation}(\pi,U,M)$

if (Terminal(s')) then $(s,a) = \text{Null}$ else $(s,a) = (s',\pi[s'])$

return a

Passive ADP Learning Curves





(a) Utility estimates for a seleted subset of states

– Large change at trial 78: 1st time agent falls into -1 terminal state

(b) Root-mean-square (RMS) error in the estimate of U(1,1)

– Averaged: over 20 runs of 100 trials each

Temporal-Difference Learning

Idea: Use observed transitions to adjust utilities of observed states

Such that: They agree with the Bellman constraint equations

$$(1,1)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (2,3)_{-.04} \mapsto (3,3)_{-.04} \mapsto (4,3)_{+1}$$

$$(1,1)_{-.04} \mapsto (1,2)_{-.04} \mapsto (1,3)_{-.04} \mapsto (2,3)_{-.04} \mapsto (3,3)_{-.04} \mapsto (3,2)_{-.04} \mapsto (4,2)_{-1}$$

$$U^{\pi}_{1}(1,3) = 0.84, \quad U^{\pi}_{1}(2,3) = 0.92, \quad U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3) = 0.88$$

Temporal-difference (TD): Use utilities difference. Learning rate α

$$U^{\pi}_{n+1}(s) = U^{\pi}_{n}(s) + \alpha(R(s) + \gamma U^{\pi}_{n}(s') - U^{\pi}_{n}(s))$$

Idea: Adust U^{π}_{n+1} towards ideal equilibrium that holds locally

Subtleties: Notice that

- Update involves successor s' only whereas Bellman involves all successors s'
- Average U^{π} still converges even for low transition probabilities to s'
- Taking function $\alpha(\nu_s)$ then $U^{\pi}(s)$ will converge itself to correct value

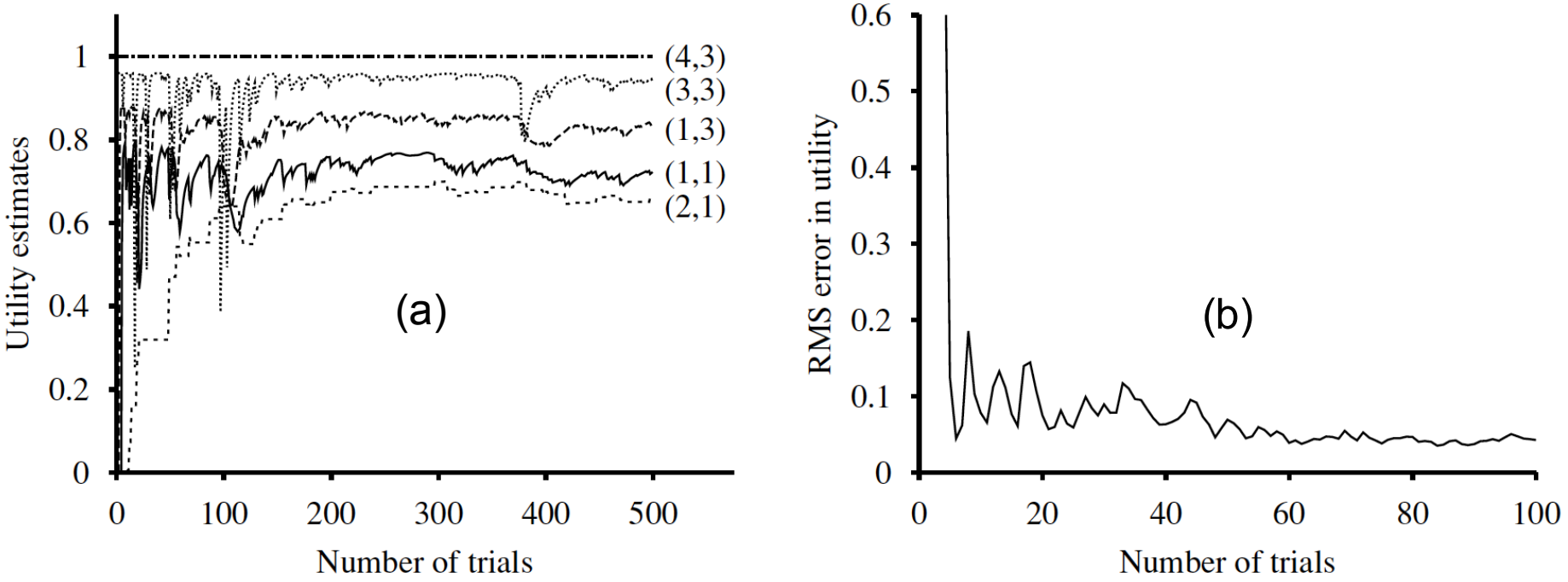
Passive TD Algorithm

function Passive-TD-Agent (Percept (s', r')) returns Action

persistent Policy π // fixed policy
persistent Table $U = \varnothing$ // utilities table
persistent Table $N_s = \varnothing$ // state-frequency table
persistent (State s, Action a, Reward r) = Null // previous s,a,r

if ($s' \notin dom(U)$) $U[s'] = r'$
if ($s \neq$ Null) {
$N_s[s]$++
$U[s] = U[s] + \alpha(N_s[s])\,(r + \gamma\, U[s'] - U[s])$ }
if (Terminal(s')) then (s,a,r) = Null else $(s,a,r) = (s', \pi[s'], r')$
return a

Passive TD Learning Curves



(a) Utility estimates for a seleted subset of states

(b) Root-mean-square error in the estimate of U(1,1)

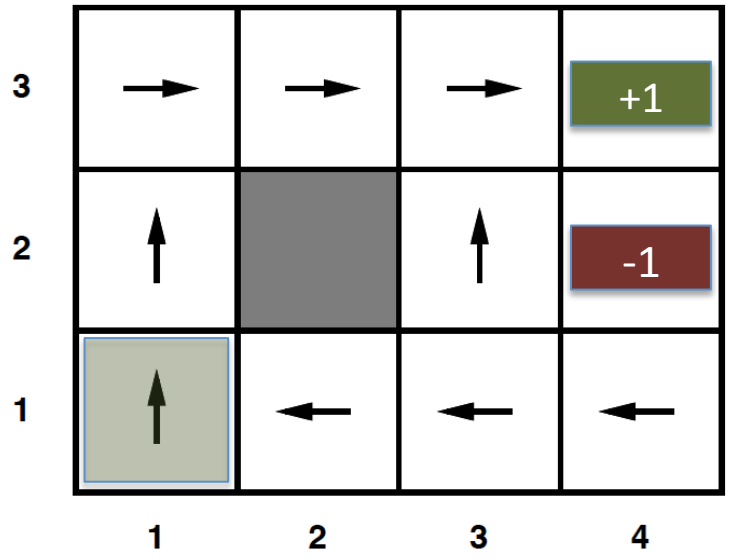
- Averaged: over 20 runs of 100 trials each

Does not learn so fast: But it does not need a transition model!

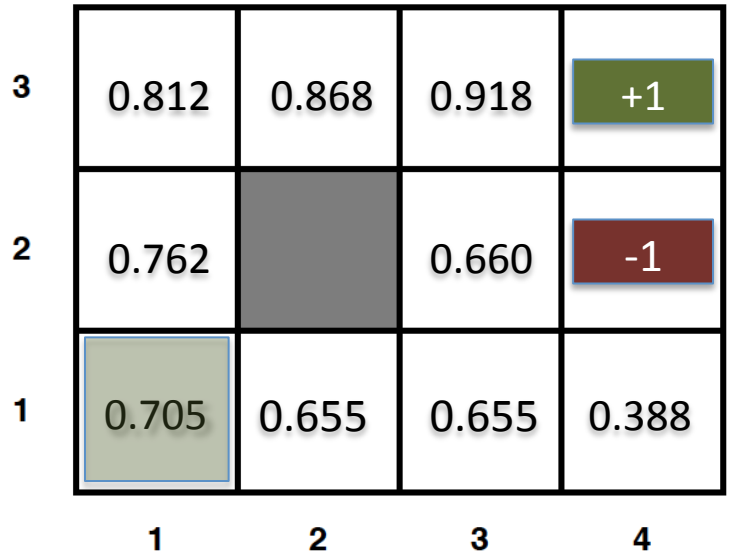
- TD is a crude: But efficient first approximation of ADP
- ADP adjustments: Result of simulated (pseudoexpereience) TD adjustments
- Prioritized sweeping: Adjust only s whose likely s' undergo large adjustments

Active Reinforcement Learning

Goal: Learn policy π



Goal: Learn utility U



Transition model $P(s' \mid s,a)$ Not known (learn as before)

Policy function $\pi(s)$ Not known (learn using exploration)

Reward function $R(s)$ Not known (use observations)

$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s,a)\, U(s')$$

States $(i,j) \in \{1..4\} \times \{1..4\} \setminus (2,2)$ Fully observable

Observations $(i,j)_R$ Reward-indexed states

Exploration

Greedy ADP agent: Uses learned model to compute utilities

- Problem: Model is different from the true environment
- Problem: The agent does not know the true environment
- Fix: Actions also contribute to learning the true model by affecting percepts

Agent must therefore: Balance between

- Exploitation: Of the model to maximize its reward (utility estimate)
- Exploration: Of various actions to maximize its long-term well being
- Pure exploration: Of no use if one never puts knowledge into practice

Bandit problems: Study if there is an optimal exploration policy

- 1-armed bandit: One slot machine. Gambler inserts coin, pulls lever, collects win
- n-armed bandit: n levers. Gambler must choose first the lever

Optimal behavior: What is exactly meant by that?

- Most definitions: Maximize the expected total reward over agent's lifetime
- Assumption: Expectation taken over all possible worlds $P(s'|s,a)$
- n-independent SMs: Possible to compute a Gittins index for each machine

Greedy in the Limit of Infinite Exploration

Optimal exploration: Difficult to solve. Greedy approximation (GLIE)

- Must try each action: In each state an unbounded number of times
- ADP agent using this scheme: Will eventually learn the true environment
- Must also eventually become greedy: So that agent's model is used

Several possible GLIE schemes:

- Simplest : Choose randomly an action a fraction 1/t of the time; otherwise greedy
- Alternative: Give weight to actions not often tried, avoiding low utility actions
- Amounts to: Optimistic prior over the possible environments.

Bellman with optimistic estimate

$$U^+(s) = R(s) + \gamma \max_a f\left(\sum_{s'} P(s'|s,a)\, U^+(s'),\ N(s,a) \right)$$

- $U^+(s)$: Optimistic estimate of the utility (expected reward to go)
- N(s,a): The number of times action a has been tried in state s
- f(u,n): The exploration function. Tradeoff between greed and curiosity

$$f(u,n) = (n < N_e)?\ R^+ :\ u \quad \text{where} \quad R^+ = \text{best reward},\ N_e = \text{fixed parameter}$$

- Use of U^+ in the RHS: Benefits of exploration propagated back

Active ADP Algorithm

function Active-ADP-Agent (Percept (s', r')) returns Action

persistent MDP M // an MDP $M = (P,R,\gamma)$

persistent Table $U = \varnothing$ // a table of utilities

persistent Table $N_{sa} = \varnothing$ // (state,action) frequencies table

persistent Table $N_{s'|sa} = \varnothing$ // s' frequency for (s,a)

persistent State $s = \text{Null}$, Action $a = \text{Null}$ // previous s and a

if ($s' \notin \text{dom}(U)$) $U[s'] = r'$; $R[s'] = r'$

if ($s \neq \text{Null}$) {

$N_{sa}[s,a]$++; $N_{s'|sa}[s',s,a]$++

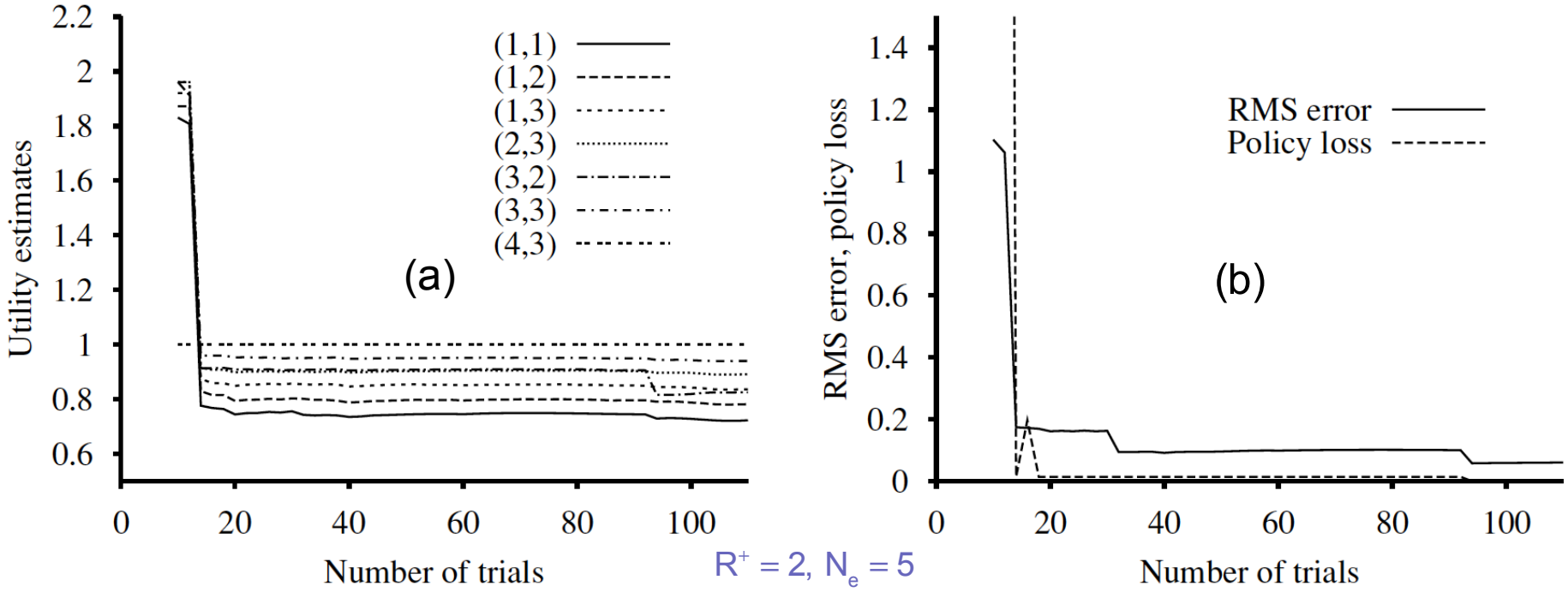
foreach (t st. $N_{s'|sa}[t,s,a] \neq 0$) $P(t \mid s,a) = N_{s'|sa}[t,s,a] / N_{sa}[s,a]$ }

$U^+ = \text{Exploratory-Value-Iteration}(P(t|s,a),U^+,f)$

if (Terminal(s')) then $(s,a) = \text{Null}$ else $(s,a) = (s',\pi[s'])$

return a

Performance of Exploratory ADP



$R^+ = 2$, $N_e = 5$

(a) Utility estimates for a seleted subset of states

- Near optimal policy: Found after just 18 trials! Fast convergence
- Utility estimates: Converge somewhat slower (stops exploring)

(b) Root-mean-square error and policy loss in the estimate of U(1,1)

- Averaged: over 20 runs of 100 trials each

Learning an Action-Utility Q-Function

Active TD agent: Has no fixed policy. Hence, it needs a model
- Model-acquisition problem: Identical to that for ADP
- Update rule: Remains unchanged. It converges to same value as ADP

Q-Learning: Alternative to active TD method
- Learns action-utility representation: Instead of learning utilities
- Q(s,a): Value of doing action a in state s: $U(s) = \max_a Q(s,a)$
- Property: TD agent using Q doesn't need a model for learning/action-selection

$$Q(s,a) = R(s) + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

- Direct use of this inductive rule: Needs a model P(s'|s,a)

TD approach however: Requires no model P(s'|s,a)
- The TD update rule: Simplifies to the following recursive equation

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

Exploratory Q-Learning Algorithm

function Q-Learning-Agent (Percept (s', r')) returns Action

persistent Function f // given exploration function (as in ADP)

persistent Table $Q = \varnothing$ // Q-values table indexed by (s,a)

persistent Table $N_{sa} = \varnothing$ // (state,action) frequencies table

persistent (State s, Action a, Reward r) = Null // previous s,a,r

if (Terminal(s)) then $Q(s,\text{None}) = r'$

if ($s \neq \text{Null}$) {

$N_{sa}[s,a]$++

$Q[s,a] = Q[s,a] + \alpha(N_{sa}[s,a])(r + \gamma \max_{a'} Q[s',a'] - Q[s,a])$ }

$(s, a, r) = (s', \arg\max_{a'} f(Q[s',a'], N_{sa}[s',a']), r')$

return a

Q-Learning and SARSA

SARSA (State-Action-Reward-State-Action) : Close relative

- Update-rule: Backs-up the Q-value of the actually taken action

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma\, Q(s',a') - Q(s,a)) \qquad \text{versus}$$

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

- Without exploration: Q-learning and SARSA are identical
- With exploration: Q-learning (takes best value) is off- while SARSA on-policy
- Q-Learning: Performs well even guided by an adversarial exploration policy
- SARSA: More realistic. Learns what actually happens

Both Q-L and SARSA learn optimal policy: For the 4x3 world

- Do so: At much slower rate than the ADP agent
- Reason: Local updates do not enforce consitency among all Q-values

Learning model/utility versus action-utility: Which is better?

- AI knowledge-based approach: Implicitly assumes learning agent's model
- Model-free approach: Simple but may be less effective for complex models

Generalized Reinforcement Learning

Utility- and Q-functions were tables: Not scalable for realistic worlds

- Backgammon and chess: Tiny subsets of realistic worlds, but 10^{40} states
- Absurd to assume: one must visit these states many times in order to play

Function approximation: One way of handling such problems

- Idea: Represent U and Q in a particular basis
- Basis or features: A set of functions $f_1,\ldots,f_n$ such that Q or U is represented as

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots + \theta_n f_n(s)$$

- RL algorithm: Learns the values for the parameters $\theta_1,\ldots,\theta_n$. $\hat{U}_\theta \simeq U$
- Enormous compression: From 10^{40} to say 20 parameters θ_i

- More importantly: Enables induct. generalization from visited to ¬visited states

GRL with Direct Utility Estimation

Features of 4x3 world: their x and y coordinates, so we have

$\hat{U}_\theta(x,y) = \theta_0 + \theta_1 x + \theta_2 y$ where $f_1(s) = x$ and $f_2(s) = y$ are linear

- Thus if $(\theta_0, \theta_1, \theta_2) = (0.5, 0.2, 0.1)$: $\hat{U}_\theta(1,1) = 0.8$
- Given a collection of trials: We obtain a set of samples for $\hat{U}_\theta(x,y)$
- Then find: Best fit of θ minimizing the squared error by linear regression
- For RL: More sense to use online algorithm to update param for each trial

Minimization: Use an error function and compute its gradient wrt θ

- Let $u_j(s)$: Observed total reward from state s onward in jth trial

$E_j(s) = (u_j(s) - \hat{U}_\theta(s))^2 / 2$

- Rate of change in θ_i is $\partial E_j / \partial \theta_i$: To move in decreasing direction update

$\theta_i \leftarrow \theta_i - \alpha \partial E_j / \partial \theta_i = \theta_i + \alpha (u_j(s) - \hat{U}_\theta(s)) \partial \hat{U}_\theta(s) / \partial \theta_i$ Widrow-Hoff or Δ rule

- For the linear function approx of $\hat{U}_\theta(s)$: We get three simple update rules

$\theta_0 \leftarrow \theta_0 + \alpha(u_j(s) - \hat{U}_\theta(s))$ $\theta_1 \leftarrow \theta_1 + \alpha(u_j(s) - \hat{U}_\theta(s))x$ $\theta_2 \leftarrow \theta_2 + \alpha(u_j(s) - \hat{U}_\theta(s))y$

- Example: $\hat{U}_\theta(1,1) = 0.8$, $u_j(1,1) = 0.4$, then $\theta_0, \theta_1, \theta_2$ are all decreased by 0.4α
- Changing θ after an observed transition: Also changes $\hat{U}_\theta$ for the next state

GRL with Direct Utility Estimation

Linear approx: In θ. Features $f_i(s)$ can be nonlinear functions

- One can include $\theta_3 f_3(x,y) = \theta_3 \sqrt{(x-x_y)^2 + (y-y_y)^2}$ measuring distance to goal

Applies also to TD learners: Adjust θ to reduce the TD

- New version of TD- and Q-learning: Equations given by

$$\theta_i \leftarrow \theta_i + \alpha\,(R(s) + \gamma\,\hat{U}_\theta(s') - \hat{U}_\theta(s))\,\partial\hat{U}_\theta(s)/\partial\theta_i \quad \text{for utilities}$$

$$\theta_i \leftarrow \theta_i + \alpha\,(R(s) + \gamma\,\max_{a'}\hat{Q}_\theta(s',a') - \hat{Q}_\theta(s,a))\,\partial\hat{Q}_\theta(s,a)/\partial\theta_i \quad \text{for Q-values}$$

- Passive TD-L: Can be shown go converge to closest approximation
- Active TD-L and nonlinear features: All bets are off. RL still a delicate art

Function approximation: Can be very helpful to learn a model

- For observable environment: Is a supervised learning problem
- Next percept: Gives the outcome state
- For partially obs environment: Learning problem much more difficult
- Inventing hidden variables and model structure: Still open problems

Policy Search

Simplest: Keep twiddling policy as long as it improves. Then stop.

- One can include $\theta_3 f_3(x,y) = \theta_3 \sqrt{(x-x_y)^2 + (y-y_y)^2}$ measuring distance to goal

Represent π: As a collection of parameterized Q-functions

$\pi(s) = \text{argmax}_a \hat{Q}_\theta(s,a)$

- Each Q-function: Can be a linear function of parameters θ
- Policy search: Adjusts parameters θ to improve the policy
- Q-learning with function approximation: Finds θ such that $\hat{Q}_\theta \simeq Q^*$ (optimal)
- The values of the two: May differ significantly
- Q-function: $\hat{Q}_\theta(s,a) = Q^*(s,a)/10$ optimal performance but not close to Q^*

Problem with above policy representation: It is discontinous in θ

- Infinitesimal change in θ: Switch from one action to another
- Gradient search: A nightmare
- Fix Softmax Stochastic Policy: Specifying a probability of action selection

$$\pi_\theta(s) = e^{\hat{Q}_\theta(s,a)} / \sum_{a'} e^{\hat{Q}_\theta(s,a')}$$

- Becomes nearly deterministic: If one action is much better than all other

Improving-Policy Methods

Deterministic policy and deterministic environment: Simplest

- Policy value $\rho(\theta)$: Expected reward-to-go when π_θ is executed
- Closed form: Policy improvement reduces to standard optimization
- Follow policy gradient vector $\nabla_\theta \rho(\theta)$: Provided $\rho(\theta)$ is differentiable
- Not closed form: Evaluate π_θ by executing it and observing accumulated reward
  - Follow empirical gradient: Evaluate change in policy value for small $\Delta\theta$

Stochastic environment: Things get more difficult

- Try hill climbing: Requires comparing $\rho(\theta)$ and $\rho(\theta+\Delta\theta)$ for small $\Delta\theta$
- Problem: Total reward on each trial may vary widely $\Rightarrow\gg$ number of trials

Stochastic policy $\pi_\theta(s,a)$: Things get also more difficult

- Possible to obtain unbiased estimate of $\nabla\rho(\theta)$: Directly from trials executed at θ
  - Non-sequential environment: R(a) obtained immediately after doing a in s_0
  - Policy value: Expected value of the reward, so we have

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0,a)\, R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0,a))\, R(a)$$

Improving-Policy Methods

Stochastic policy $\pi_\theta(s,a)$: Things get also more difficult

- Possible to obtain unbiased estimate of $\nabla\rho(\theta)$: Directly from trials executed at θ

$$\nabla_\theta \rho(\theta) = \nabla_\theta \sum_a \pi_\theta(s_0,a)\, R(a) = \sum_a (\nabla_\theta \pi_\theta(s_0,a))\, R(a)$$

- Trick: Approximate tue gradient by a sum of action-selection-probability gradient

$$\nabla_\theta \rho(\theta) = \sum_a \pi_\theta(s_0,a)\, \frac{(\nabla_\theta \pi_\theta(s_0,a))\, R(a)}{\pi_\theta(s_0,a)} \simeq \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s_0,a_j))\, R(a_j)}{\pi_\theta(s_0,a_j)}$$

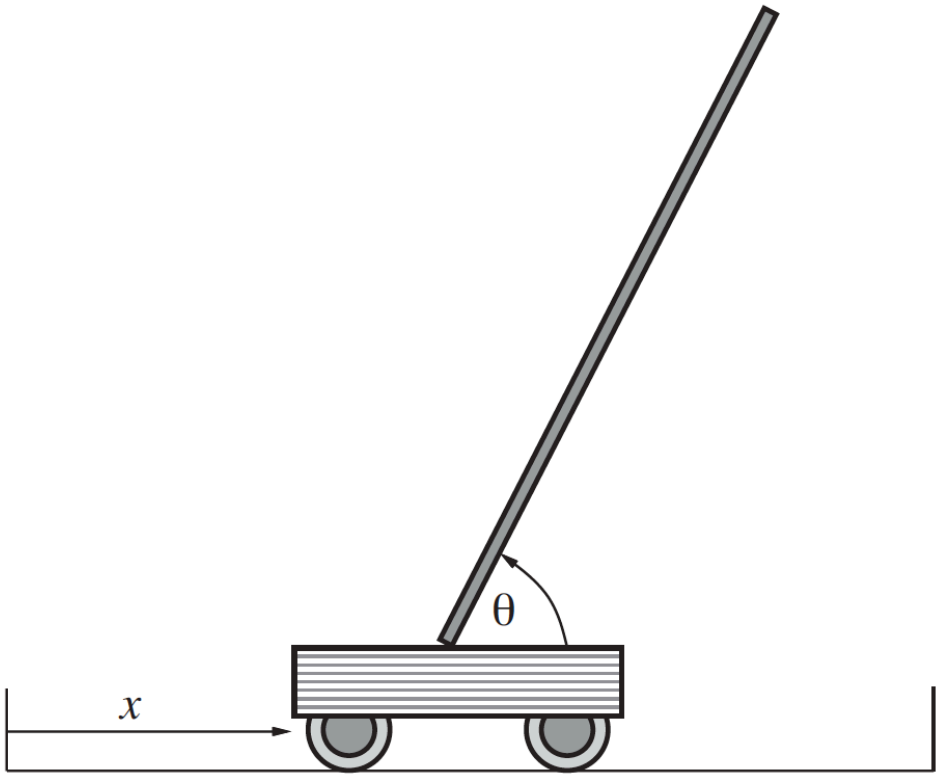
- Sequential env: $\forall$state s $\forall$trial j. a_j is executed in j and $R_j(s)$ is reward-to-go

$$\nabla_\theta \rho(\theta) \simeq \frac{1}{N} \sum_{j=1}^{N} \frac{(\nabla_\theta \pi_\theta(s,a_j))\, R_j(s)}{\pi_\theta(s,a_j)}$$

resulting alg called REINFORCE

- REINFORCE: Much more effective than hill clibing using lots of trials at each θ
- Correlated sampling: Compare policies on hands generated in advance (bridge)
- PEGASUS: Uses policy search with correlated samples

Applications of RL: Inverted Pendulum



Problem:

Control the position of x

so that the pole stays at

$\theta \simeq \pi/2$ (roughly upright)

within the track limits

State is continuous:

$\mathbf{x} = (x, \theta, \dot{x}, \dot{\theta})$

Actions are discrete:

A = {jerk-left, jerk-right}

bang-bang control

Boxes Algorithm: Discretized state space into boxes

- Negative reinforcement: Applied when poll fell or car outside track range
- Discretization: Problems. Adaptive partitioning according to reward variation
- Continuous state: Nonlinear function approximation with neural networks

Applications of RL: Helicopter Flight

Superimposed time lapses:
A difficult nose-in-circle maneuver

Controller:
Far exceeded Human expert pilot using remote control

Policy search: As well as Pegasus algorithm with correlated samples

- Simulator: Developed to observe effects of control manipulation in real helicopter
- Policy search: Simulator run overnight and policies were compared