Reinforcement Learning

Kinds of Learning

- we know the correct output label for each example: supervised learning
  - e.g., a chess-playing agent that is told explicitly the “correct” move for each position
- what about when agent only receives a reward (or penalty) at the end of a sequence of actions?
  - e.g., chess-playing agent “checkmates” at end of game
  - temporal credit assignment problem: which was the good move?

Reinforcement Learning

- how does an agent learn when given:
  - no (or limited) model of environment?
  - no utility function?
- use rewards or reinforcement to learn agent function

Agent Types

- utility-based agent
  - learns a utility function on states and uses this to select actions that maximize utility of outcome (i.e., next state)
- Q-learning agent
  - learns an **action-utility function**: expected utility of taking action *a* in state *s*
- reflex agent
  - learns a policy that maps directly from states to actions

Utility-based vs. Q-learning agents

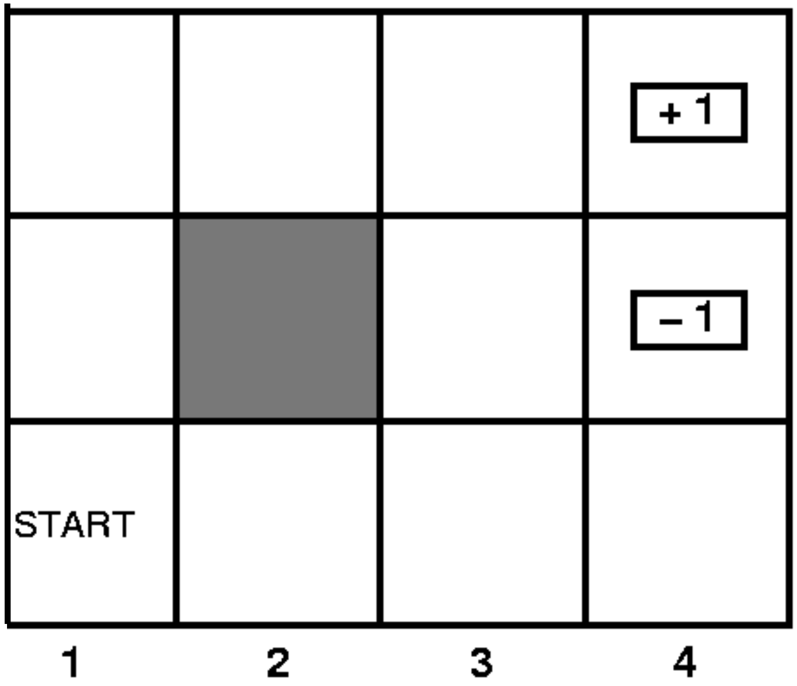
- utility-based agent
  - has to know the state to which its action will lead to determine utilities
  - therefore needs a model of the environment
- Q-learning agent
  - can compare the values of its choices without knowledge of the outcome state
  - therefore it doesn’t need a model of the environment

Characterizing the Learning Task

- Environment: known or unknown
  - does agent know effects of actions?
  - does agent have a model of environment?
  - (we will assume accessible environment)
- Learning Type: passive or active
  - passive: agent has a fixed policy (only learns how "good" each state or action is)
  - active: agents has to learn **what** to do
- Rewards: terminal only or non-terminal

Passive learning in a fully observable environment

- policy π is fixed
- want to learn how good $\pi(s)$ is, i.e., $U^{\pi}(s)$
- similar to policy evaluation
- main difference: agent does not know:
  - transition model $P(s'|s,a)$
  - the reward function $R(s)$



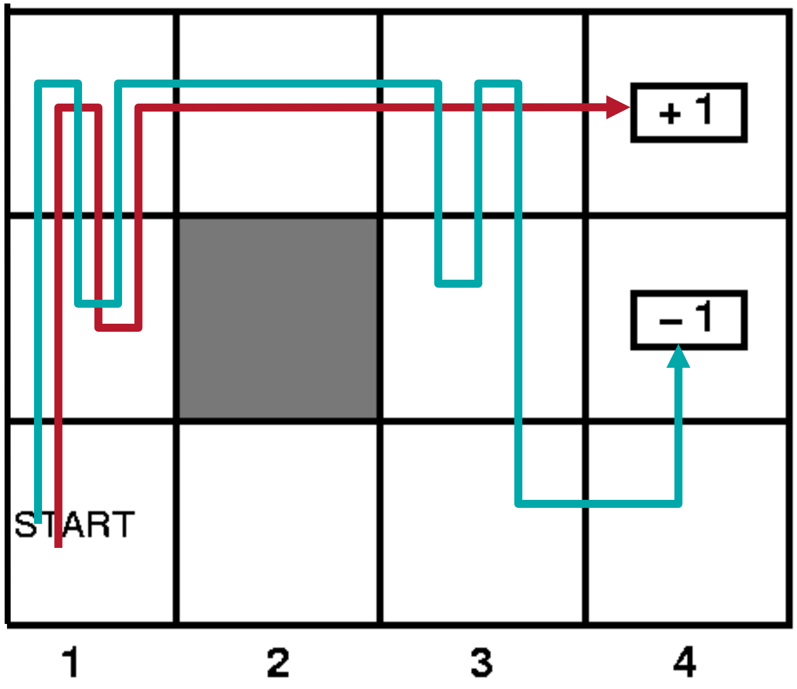
Direct Utility Estimation (DUE)

- recall: $U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^{t} R(s_{t}) \mid \pi, s_{0} = s]$

- so, agent can execute a series of trials, using each to obtain a sample of the reward-to-go for each state visited

DUE

- maintain a running average of the reward-to-go values
- after infinitely many trials, the averages will converge to true expected values

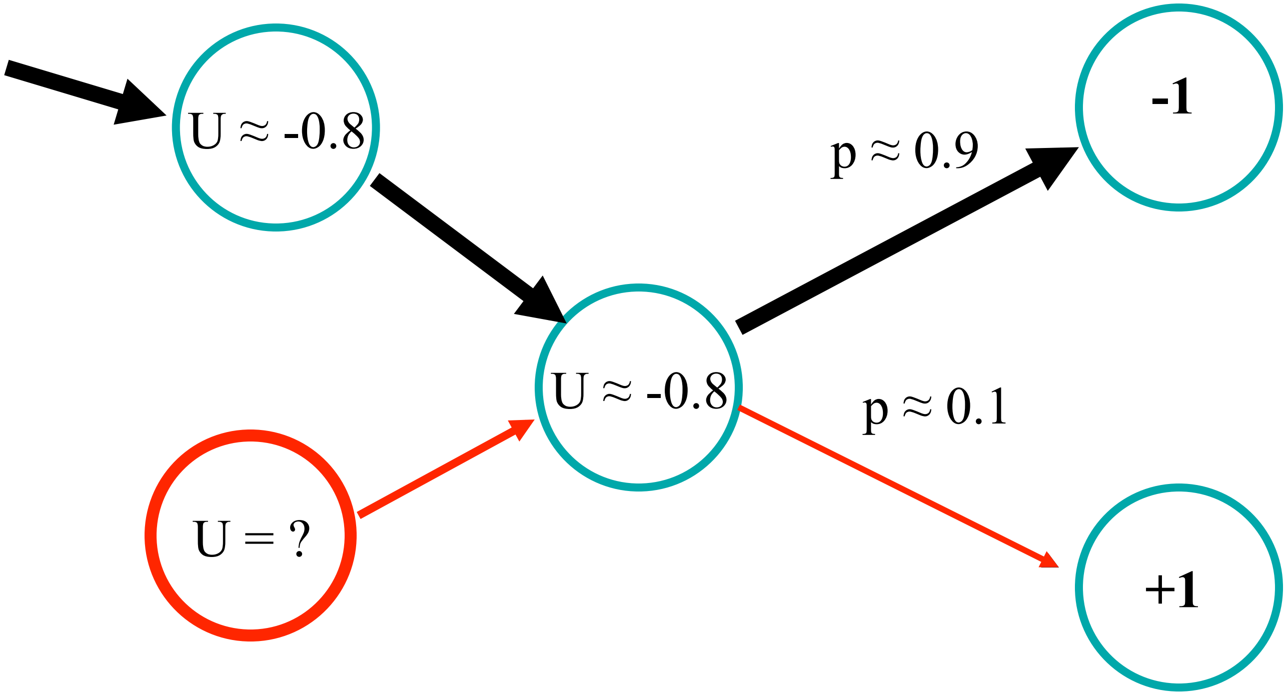


$(1,1)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (1,2)_{-.04} \rightarrow (1,3)_{-.04} \rightarrow (2,3)_{-.04} \rightarrow (3,3)_{-.04} \rightarrow (4,3)_{+1}$

DUE Algorithm

```
add $s$ to *percepts*
if TERMINAL?[$s$] then
    *reward-to-go* ← 0
    for each $s_i$ in *percepts* (starting at end) do
        *reward-to-go* ← *reward-to-go* + REWARD[$s_i$ ]
        $U[s_i]$ ←RUNNING_AVG ($U[s_i]$, *reward-to-go*, $N[s_i]$)
        increment $N[s_i]$
```

How fareth DUE?



- ignores all of its acquired experience whenever it encounters a new state… leads to slow convergence

But you can do better!

- recall: utility of states are not independent
- for a fixed policy:

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

Adaptive Dynamic Programming

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

- learn transition model of the environment *$P(s'|s,\pi(s))$* and observe rewards *$R(s)$*
- solve MDP using linear algebra
  - can solve the set of linear equations
  - or use a modified variant of policy iteration
- alas, intractable for large state spaces

ADP algorithm

```
// $s'$ , $r'$  current state, reward,  $\pi$, a fixed policy
// $N_{sa,}$ table of frequencies for state-action pairs, initially zero
// $N_{s'|sa}$, table of outcome frequencies given state-action pairs
if $s'$ is new then $U[s'] \leftarrow r'$ ; $R[s'] \leftarrow r'$
if $s$ is not null then
    increment $N_{sa}[s,a]$ and $N_{s'|sa}[s',s,a]$
    for each $t$ such that $N_{s'|sa}[t,s,a]$ is nonzero do
        $P(t\,|s,a) \leftarrow N_{s'|sa}[t,s,a]\, /\, N_{sa}[s,a]$
$U \leftarrow$ POLICY-EVALUATION$(\pi,U,mdp)$
if TERMINAL($s'$) then $s,a \leftarrow$ null else $s,a \leftarrow s', \pi(s')$
return $a$
```

Temporal difference learning

- approximate constraint equations without solving them for all states:
- use observed transitions to adjust values of observed states so that they agree with the constraint equations

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

Passive TD Update Algorithm

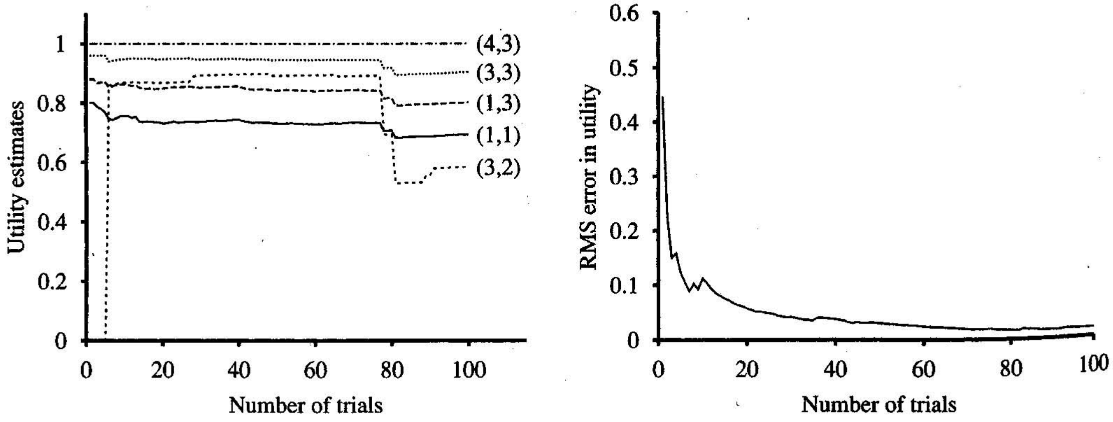
```
// s, a, r, previous state, action and reward
// s' , r'  current state, reward

if s' is new then U[s'] ← r'
if s is not null then
    increment N[s]
    $U[s] \leftarrow U[s] + \alpha(N[s])\ (r + \gamma U[s'] - U[s])$
if TERMINAL(s') then s, a, r ← null
else s, a, r ← s', $\pi$[s'], r'
return a
```

TD vs. ADP

- ADP updates utility estimates to make each state “agree” with successors:
  as many adjustments as necessary to restore consistency
- TD updates estimate only for *observed* successor state:
  a single adjustment per iteration
- isn’t this wrong? what if we adjust for a low probability outcome?
- TD is a crude but efficient first approximation for ADP

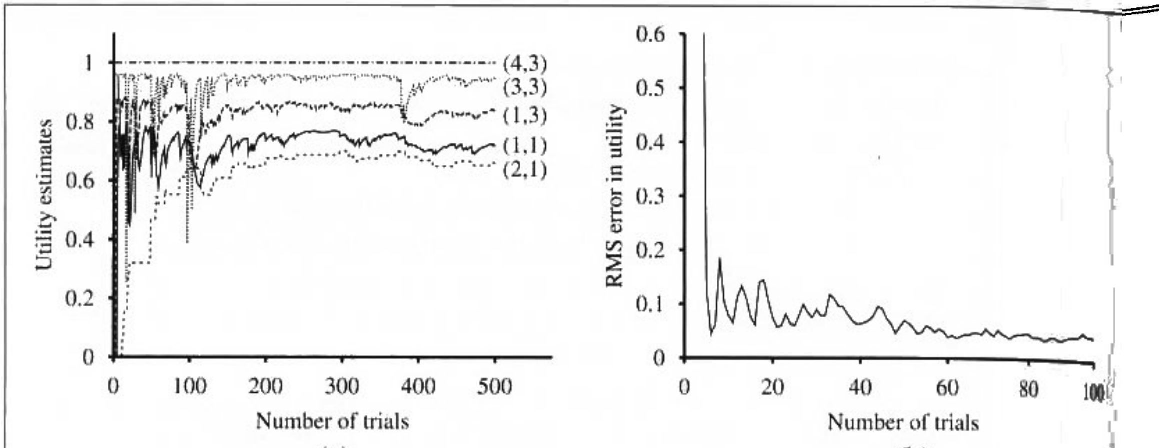
ADP vs. TD Performance



(a)

(b)

ADP



(a)

(b)

TD

Homework

- Reinforcement Learning Exercise
- Do question #1 now, question #2 after Friday’s lecture