Reinforcement Learning for Active Agents

Active Reinforcement Learning

- now need to learn model for all actions, not just for a fixed policy
- utilities obey Bellman equations:

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

can solve using value iteration
or policy iteration seen before

TD Update Algorithm for Active Agent
What has to change?

// *s, a, r,* previous state, action and reward

// *s' , r'* current state, reward

____

if *s'* is new then U[s'] $\leftarrow$ *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* $\leftarrow$ null

else *s, a, r* $\leftarrow$ *s',* ____________________, *r'*

return a

TD Update Algorithm for Active Agent

What has to change?

// *s, a, r,* previous state, action and reward

// *s' , r'* current state, reward

UPDATE MODEL (P,s,a,s')

if *s'* is new then U[s'] $\leftarrow$ *r'*

if *s* is not null then

  increment N[s]

  U[s] $\leftarrow$ U[s] + α(N[s]) (r + γU[s'] - U[s])

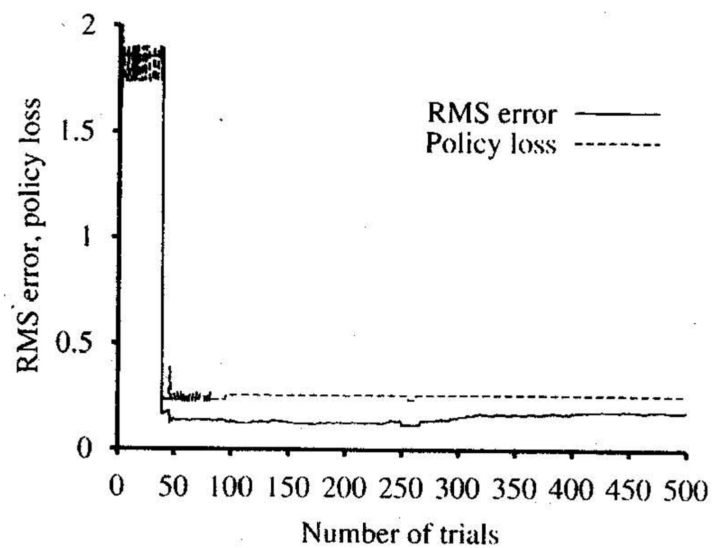
if TERMINAL?[s'] then *s, a, r* $\leftarrow$ null

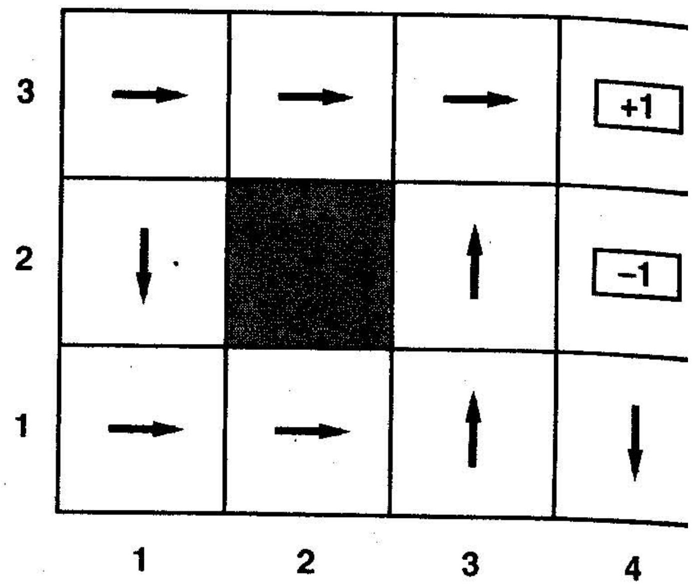
else *s, a, r* $\leftarrow$ *s', CHOOSE ACTION(P,U,s), r'*

return a

How to choose action?

- greedy agent: pick whichever action has highest expected utility
  + gives us best expected score
  – doesn't give agent a chance to explore





Optimistic Prior

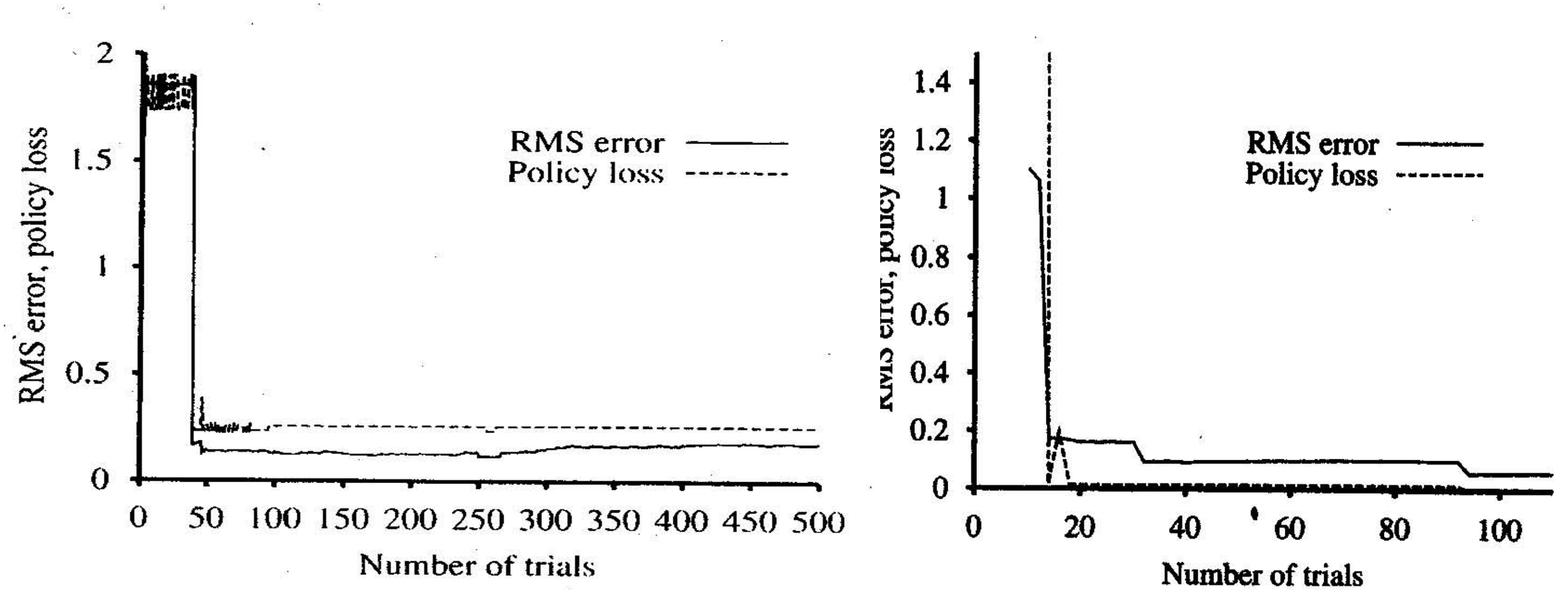
- modified constraints that assume existence of rewards in unexplored states

$$U^+(s) \leftarrow R(s) + \gamma \max_a f(\Sigma_{s'} P(s'|s,a)U^+(s'), N(s,a))$$

Exploration Function *f(u,n)*

- increasing in *u,* decreasing in *n*
- why is U^+ rather than plain U used on the RHS?
- if only U were used:
  - unexplored states would be valued
  - but not explored states **leading to** unexplored states

Performance of exploratory ADP



(a) (b)

Greedy ADP

Action-Value Function

- value of doing action a in state s is $Q(s,a)$
- then $U(s) = max_a Q(s,a)$
- constraint equation at equilibrium:

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

- can apply constraint equation for iterative update using ADP
- but this means we need to learn model, P

TD Q-learning

- learn from experience:

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

- also known as SARSA:
  State(t), Action(t,) Reward, State(t+1), Action(t+1)
  - doesn't need *P*
  - but again, exploration function is very important
  - incorporate optimistic priors into Q-value estimates

Exploratory Q-Learning-Agent

// s, a, r, previous state, action, and reward, initially null

// s' is current state, r' is reward signal

if TERMINAL?[s'] **then** Q[s,None] $\leftarrow$ r'

if s not null **then**

increment N_{sa}[s,a]

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

s, a, r $\leftarrow$ s', $\mathrm{argmax}_{a'}$ f(Q[a',s'], N[s',a']), r'

return a

Samuel's checkers

- **Scoring function:** based on the position of the board at any given time, tries to measure the chance of winning for each side at the given position.
- Program chooses its move based on a minimax strategy
- **Self-improvement:** Remembering every position it had already seen, along with the terminal value of the reward function. It played thousands of games against itself as another way of learning.
- **Success**: First to play any board game at a relatively high of level (by mid-1970s) -- earliest successful machine learning research

TD-Gammon

Tesauro, 1992

- learned to play backgammon extremely well, using a neural network function approximator trained by TD methods
- actually influenced play of expert humans!

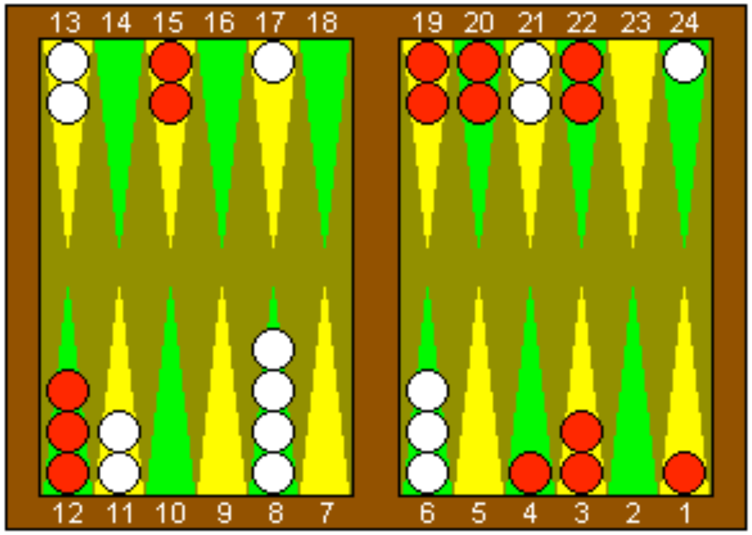


Figure 3. A complex situation where TD-Gammon's positional judgment is apparently superior to traditional expert thinking. White is to play 4-4. The obvious human play is 8-4*, 8-4, 11-7, 11-7. (The asterisk denotes that an opponent checker has been hit.) However, TD-Gammon's choice is the surprising 8-4*, 8-4, 21-17, 21-17! TD-Gammon's analysis of the two plays is given in Table 3.

OBELIX
Mahadevan and Connell, 1991

- robot performed 3 behaviours in priority order:
  - unwedge (if stuck)
  - push box
  - find box
- huge state space!

Generalization in RL

- if action *a* is good in state *i*
  then for all states *j* such that $j \approx i$
  action *a* is *probably good* in state *j*

How do we generalize?

- naïve solution:
  - course discretization of state space
- elegant solutions:
  - update neighbouring states based on similarity
  - statistical clustering techniques