CS 730/830: Intro AI

Solving MDPs

Markov Decision Process (MDP)

initial state: s_0
transition model: $T(s, a, s') =$ probability of going from s to s' after doing a.
reward function: $R(s)$ for landing in state s.
terminal states: sinks = absorbing states (end the trial).

objective:

total reward: reward over (finite) trajectory:
$R(s_0) + R(s_1) + R(s_2)$
discounted reward: penalize future by γ:
$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) \ldots$

find:

policy: $\pi(s) = a$
optimal policy: π^*
proper policy: reaches terminal state

What to do?

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s') U^{\pi^*}(s')$$

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

The key:

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

(Richard Bellman, 1957)

Repeated Bellman updates:

Repeat until happy
  for each state s
    $U'(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$
  $U \leftarrow U'$

For infinite updates everywhere, guaranteed to reach equilibrium.

Equilibrium is unique solution to Bellman equations!

asychronous works: converges if every state updated infinitely often (no state permanently ignored)

Stopping

$||U_i - U_{i-1}||$ = max difference between corresponding elts

$U^* = U^{\pi^*}$

if $||U_i - U_{i-1}||\gamma/(1-\gamma) < \epsilon$ then $||U_i - U^*|| < \epsilon$

Stopping

$||U_i - U_{i-1}|| =$ max difference between corresponding elts

$U^* = U^{\pi^*}$

if $||U_i - U_{i-1}||\gamma/(1-\gamma) < \epsilon$ then $||U_i - U^*|| < \epsilon$

if $||U_i - U^*|| < \epsilon$ then $||U^{\pi_i} - U^{\pi^*}|| < 2\epsilon\gamma/(1-\gamma)$

Stopping

$||U_i - U_{i-1}||$ = max difference between corresponding elts

$U^* = U^{\pi^*}$

if $||U_i - U_{i-1}||\gamma/(1-\gamma) < \epsilon$ then $||U_i - U^*|| < \epsilon$

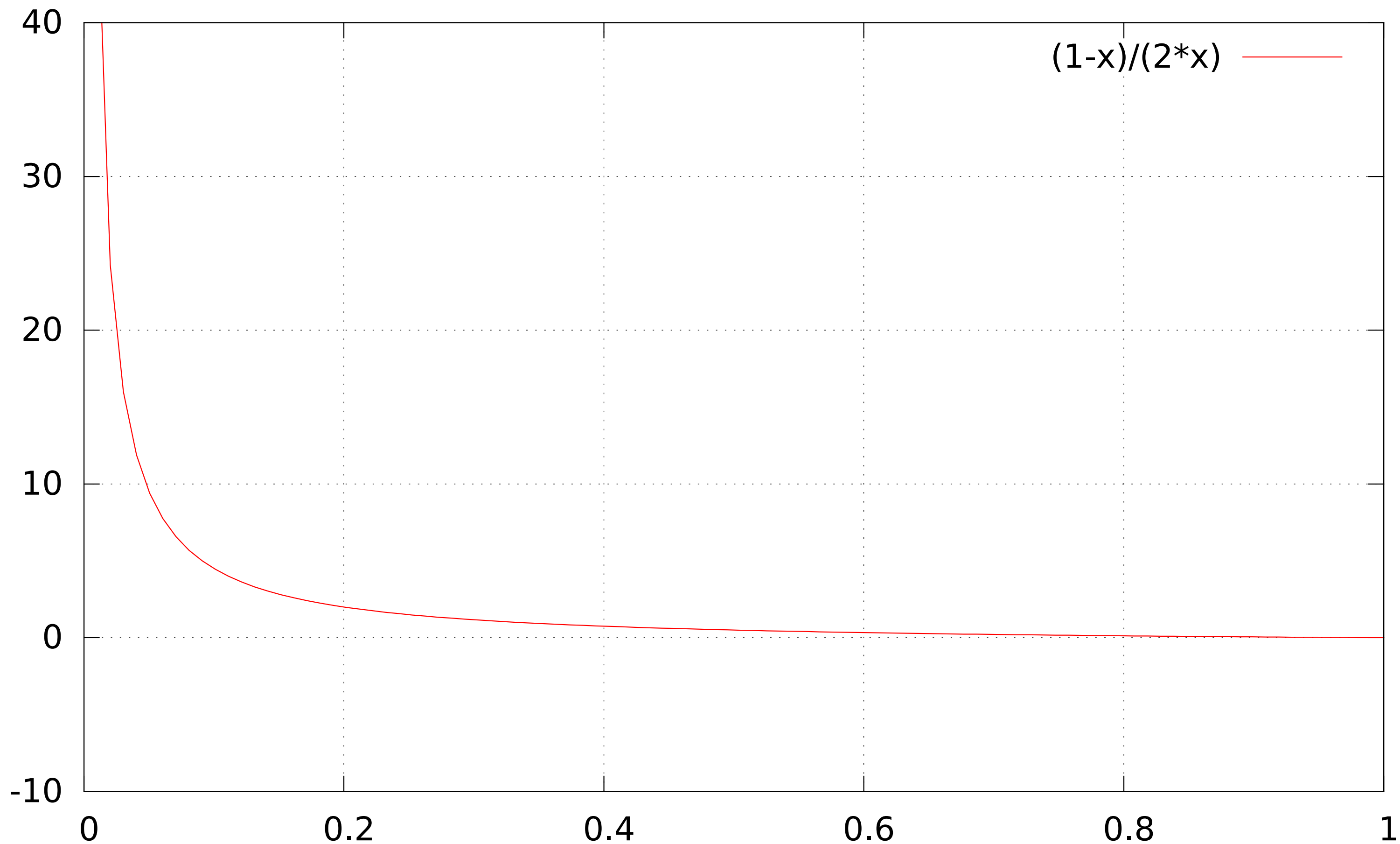
if $||U_i - U^*|| < \epsilon$ then $||U^{\pi_i} - U^{\pi^*}|| < 2\epsilon\gamma/(1-\gamma)$

loss $< \frac{2(maxUpdate)\gamma}{1-\gamma}$

$maxUpdate > \frac{loss(1-\gamma)}{2\gamma}$

Stopping

$$maxUpdate > \frac{loss(1-\gamma)}{2\gamma}$$



concentrate updates on states whose value changes!

to update state s with change δ in $U(s)$:
  update $U(s)$
  priority of $s \leftarrow 0$
  for each predecessor s' of s:
    priority $s' \leftarrow$ max of current and $\max_a \delta \hat{T}(s', a, s)$

Stochastic Shortest Path Problems

- minimize sum of action costs
- all action costs ≥ 0
- non-empty set of (absorbing zero-cost) goal states
- there exists at least one proper policy

proper policy: eventually brings agent to goal from any state with probability 1

Real-time Dynamic Programming (RTDP)

which states to update?

initialize U to an upper bound
do trials until happy:
- $s \leftarrow s_0$
- until at a goal:
  - $a, u_a \leftarrow \arg\min_a c(s,a) + \sum_{s'} T(s,a,s')U(s')$
  - $U(s) \leftarrow u_a$
  - $s \leftarrow$ pick among s' weighted by $T(s,a,s')$

states that agent is likely to visit under current policy
nice anytime profile
in practice, do updates backward from end of trajectory

convergence guaranteed by optimism.

Break

- asst 9
- project: start small, paper is what counts
- wildcard vote Thursday

Upper Confidence Bounds on Trees (UCT, 2006)

on-line action selection

Monte Carlo tree search (MCTS)
selection, expansion, roll-out, update

$$W(s,a) = \text{total reward}$$
$$N(s,a) = \text{number of times } a \text{ tried in } s$$
$$N(s) = \text{number of times } s \text{ visited}$$
$$Z(s,a) = \frac{W(s,a)}{N(s,a)} + C\sqrt{\frac{\log N(s)}{N(s,a)}}$$

roll-out policy
add one node after each roll-out
consistent!

AlphaGo

game logs: rollout policy, SL policy
self-play: policy network, value network
AlphaGo: MCTS using policy and value networks
AlphaZero: all self-play!

Policy Iteration

repeat until π doesn't change:
  given π, compute $U^\pi(s)$ for all states
  given U, calculate policy by one-step look-ahead

If π doesn't change, U doesn't either.
We are at an equilibrium ($=$ optimal π)!

Policy Evaluation

computing $U^\pi(s)$:

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

Policy Evaluation

computing $U^\pi(s)$:

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

linear programming (N^3) or

Policy Evaluation

computing $U^\pi(s)$:

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

linear programming (N^3) or simplified value iteration:

do a few times:

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

(simplified because we are given π, no max over a)

Summary of MDP Solving

- value iteration: compute U^{π^*}
  - prioritized sweeping
  - RTDP
  - (UCT)
- policy iteration: compute U^{π} using
  - linear algebra
  - simplified value iteration
  - a few updates (modified PI)

MDP Extras

Adaptive Dynamic Programming

'model-based'. active vs passive

learn T and R as we go, get π using MDP methods (eg, VI or PI)

example with VI:

Until *max-update* is small enough
  for each state s
    $U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s')$

$$\pi(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')U(s')$$

Exploration vs Exploitation

problem:

Exploration vs Exploitation

problem: greedy (local minima)

'multi-armed bandit' problem

random action with probability $\frac{1}{N}$

or even something like

$$U^+(s) \leftarrow R(s) + \gamma \max_a f\left(\sum_{s'} T(s,a,s')U^+(s'), N(a,s)\right)$$

where $f(u,n) = R_{\max}$ if $n < k$, u otherwise

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

Q-Learning

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

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

Given experience $\langle s, a, s', r \rangle$:

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

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

Given experience $\langle s, a, s', r \rangle$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(error)$$

Q-Learning

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

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

Given experience $\langle s, a, s', r \rangle$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(error)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha(sensed - predicted)$$

Q-Learning

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

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

Given experience $\langle s, a, s', r \rangle$:

$$\begin{aligned}
Q(s,a) &\leftarrow Q(s,a) + \alpha(\mathit{error}) \\
Q(s,a) &\leftarrow Q(s,a) + \alpha(\mathit{sensed} - \mathit{predicted}) \\
Q(s,a) &\leftarrow Q(s,a) + \alpha(\gamma(r + \max_{a'} Q(s',a')) - Q(s,a))
\end{aligned}$$

Q-Learning

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

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

Given experience $\langle s, a, s', r \rangle$:

$$\begin{aligned}
Q(s,a) &\leftarrow Q(s,a) + \alpha(\mathit{error}) \\
Q(s,a) &\leftarrow Q(s,a) + \alpha(\mathit{sensed} - \mathit{predicted}) \\
Q(s,a) &\leftarrow Q(s,a) + \alpha(\gamma(r + \max_{a'} Q(s',a')) - Q(s,a))
\end{aligned}$$

$\alpha \approx 1/N$?
policy: choose random with probability $1/N$?

RL Summary

Model known (solving MDP):

- value iteration
- policy iteration: compute U^π using
  - linear algebra
  - simplified value iteration
  - a few updates (modified PI)

Model unknown (RL):

- ADP using
  - value iteration
  - a few updates (eg, prioritized sweeping)
- Q-learning

Function Approximation

$$\hat{U}(s) = \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots$$

Function Approximation

$$\begin{aligned}\hat{U}(s) &= \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \\ \hat{U}(x,y) &= \theta_0 + \theta_1 x + \theta_2 y\end{aligned}$$

Function Approximation

$$\begin{aligned}\hat{U}(s) &= \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \\ \hat{U}(x,y) &= \theta_0 + \theta_1 x + \theta_2 y\end{aligned}$$

given sample u at $s = x, y$, want update to decrease error:

$$E = \frac{(\hat{U}(s) - u)^2}{2}$$

Function Approximation

$$\hat{U}(s) = \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots$$
$$\hat{U}(x, y) = \theta_0 + \theta_1 x + \theta_2 y$$

given sample u at $s = x, y$, want update to decrease error:

$$E = \frac{(\hat{U}(s) - u)^2}{2}$$
$$\theta_i \leftarrow \theta_i - \alpha \frac{\delta E}{\delta \theta_i}$$

Function Approximation

$$\begin{aligned}\hat{U}(s) &= \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \\ \hat{U}(x,y) &= \theta_0 + \theta_1 x + \theta_2 y\end{aligned}$$

given sample u at $s = x, y$, want update to decrease error:

$$\begin{aligned}E &= \frac{(\hat{U}(s) - u)^2}{2} \\ \theta_i &\leftarrow \theta_i - \alpha \frac{\delta E}{\delta \theta_i} \\ \theta_i &\leftarrow \theta_i - \alpha(\hat{U}(s) - u)\frac{\delta \hat{U}(s)}{\delta \theta_i}\end{aligned}$$

Function Approximation

$$
\begin{aligned}
\hat{U}(s) &= \theta_0 f_0(s) + \theta_1 f_1(s) + \theta_2 f_2(s) + \ldots \\
\hat{U}(x, y) &= \theta_0 + \theta_1 x + \theta_2 y
\end{aligned}
$$

given sample u at $s = x, y$, want update to decrease error:

$$
\begin{aligned}
E &= \frac{(\hat{U}(s) - u)^2}{2} \\
\theta_i &\leftarrow \theta_i - \alpha \frac{\delta E}{\delta \theta_i} \\
\theta_i &\leftarrow \theta_i - \alpha (\hat{U}(s) - u) \frac{\delta \hat{U}(s)}{\delta \theta_i}
\end{aligned}
$$

in other words, the updates are:

$$
\begin{aligned}
\theta_0 &\leftarrow \theta_0 - \alpha (\hat{U}(s) - u) \\
\theta_1 &\leftarrow \theta_1 - \alpha (\hat{U}(s) - u) x \\
\theta_2 &\leftarrow \theta_2 - \alpha (\hat{U}(s) - u) y
\end{aligned}
$$

Deep RL

How to choose features? Learn them!

- deep Q-learning (DQN): eg, backgammon, Atari games
  use mini-batches to try to avoid divergence
- value approximation: eg, Go outcome
  also, move probability

EOLQs

- What question didn't you get to ask today?
- What's still confusing?
- What would you like to hear more about?

Please write down your most pressing question about AI and put it in the box on your way out.

Thanks!