Reinforcement learning

Chapter 20, Sections 1–7

Outline

◇ Settings

◇ Passive learning

◇ Active learning

◇ Q-learning

◇ Generalization

Settings

Fully observable vs. partially observable
Known model vs. unknown model
Passive observation vs. active exploration

Agent designs
- Direct policy (stochastic)—adjust $P(a, i)$
- Q-function—adjust $Q(a, i)$
- Model-based—adjust $U(i)$, use lookahead

Learning methods
- "LMS"—estimate U or Q from observed sum of rewards
- Adaptive DP—learn model, solve the MDP
- TD—adapt U towards restoring local consistency

LMS update

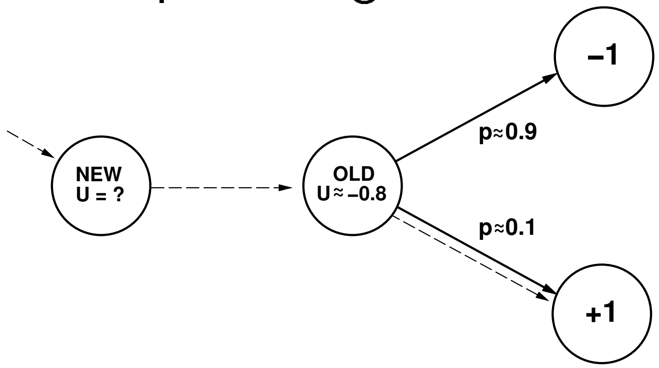
Observations are sequences of states + rewards:

$$(1,1)\rightsquigarrow(1,2)\rightsquigarrow(1,3)\rightsquigarrow(2,3)\rightsquigarrow(3,3)\rightsquigarrow(4,3)\ \underline{+1}$$
$$(1,1)\rightsquigarrow(1,2)\rightsquigarrow(1,1)\rightsquigarrow(1,2)\rightsquigarrow(1,3)\rightsquigarrow(2,3)\rightsquigarrow(1,3)\rightsquigarrow(2,3)\rightsquigarrow(3,3)\rightsquigarrow(4,3)\ \underline{+1}$$
$$(1,1)\rightsquigarrow(2,1)\rightsquigarrow(1,1)\rightsquigarrow(1,2)\rightsquigarrow(1,3)\rightsquigarrow(2,3)\rightsquigarrow(3,3)\rightsquigarrow(3,2)\rightsquigarrow(4,2)\ \underline{-1}$$

For each sequence and each state i in the sequence,
sample value for $U(i)$ = actual sum of rewards until termination

Reinforcement learning reduced to (online) supervised learning of U
(classical LMS algorithm is online linear regression)

Problem: ignores relationships among states $\Rightarrow$ slow convergence



Adaptive dynamic programming

After each transition $i \rightsquigarrow j$
  update the model and re-solve the MDP

Reinforcement learning reduced to supervised learning of M

Makes best possible use of each experience
Very expensive computationally—MDP solution mostly unchanged

Temporal difference learning

Bellman equation for passive learning: $U(i) = R(i) + \Sigma_j M_{ij} U(j)$

TD update: reduce error <u>on this transition</u>:

$$U(i) \leftarrow U(i) + \alpha(R(i) + U(j) - U(i))$$

Stochastic convergence: α decays appropriately
$\Rightarrow U(i)$ values converge to correct solution

One TD update per transaction $\Rightarrow$ slow convergence

Better methods (eg GPS) do a few high-impact updates per transition

Active learning

Experience is generated by the agent's own choices
What to do? Exploitation vs. exploration

Greedy selection: choose action maximizing $\Sigma_j M^a_{ij} U(j)$
Leads to getting "stuck in a rut"

Convergence requires exploration of all states
1) randomize a decreasing fraction ϵ of actions
2) Boltzmann exploration: probability of doing a decreases exponentially with relative value

(Exploration = MDP whose states are distributions over MDPs
Analytical solutions only for some "bandit problems")

TD update rule remains unchanged for the active case:
stochastic sampling approximation to M^a_{ij}

Q-learning

$Q(a,i) =$ value of doing a in i; $U(i) = \max_a Q(a,i)$

Bellman equation at equilibrium:

$$Q(a,i) = R(i) + \textstyle\sum_j M_{ij}^a \max_{a'} Q(a',j)$$

Q-learning update on seeing $i \rightsquigarrow j$:

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

Does not require a model for learning or decision making

Generalization

Large/infinite state spaces $\Rightarrow$ cannot visit every state

Typically use parametric function approximator for U or Q

E.g., linear combination of features $U_{\mathbf{w}}(i) = \mathbf{w} \cdot \mathbf{f}(i)$
or neural network or CMAC etc.

Update as steepest descent:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha[r + U_{\mathbf{w}}(j) - U_{\mathbf{w}}(i)]\nabla_{\mathbf{w}} U_{\mathbf{w}}(i)$$

Convergence with TD updates only for the linear approximator