Making Complex Decisions

Value Iteration and Policy Iteration

By Jayna Leone

OVERVIEW

- How to calculate an optimal policy π^* using value iteration and policy iteration
- Advantages/Disadvantages of each algorithm

UTILITIES OF STATES

- Calculate the utility of each state – or the expected utility of the state sequences that might follow it*
- Therefore, the sequence of following states depends on the policy that is executed
- Start by defining utility U^{π} (s) with respect to policy π.
- Utility state is the expected sum of discounted rewards if the agent executes an optimal policy

---

- Relationship between the utility of current state *s* and utility of its neighbors
- U(s) is immediate reward for that state plus expected discounted utility of the next state
- Defined by the Bellman equation*

$$U(1,1) = -0.04 + \gamma \max\{ 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1),\ 0.9U(1,1) + 0.1U(1,2),\ 0.9U(1,1) + 0.1U(2,1),\ 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\}$$

VALUE ITERATION

- Bellman equation is basis of value iteration algorithm for solving MDPs
- *n* states, *n* equations, *n* unknowns per equation
- NOT linear equations because “max” operator not linear
- How do we solve these ?

- Solve with value iteration
  - Start with arbitrary values for utilities of the states
  - Calculate the right-hand side of equation
  - Plug into left-hand side of equation – updating the utility of each state from the utilities of neighbors
  - Repeat until reach equilibrium
- Iterations are called Bellman updates*
- Propagates information thru the states space by means of local updates

CONVERGENCE

- Contraction: function of one argument that, when applied to 2 different inputs, produces 2 output values that are “closer together” by at least some constant amount
- Properties: 1 fixed point, value of output gets closer to fixed point and reaches it at its limit

- Apply convergence idea to Bellman update*
- Contraction by the factor of γ
- Error is reduced by factor of γ each iteration and rate of convergence = γ

- When do you stop iterating?

  – Know how much error is reduced, and at what rate iterations converge
  – Know that utilities of states are bounded by plus/minus $R_{max}/(1-\gamma)$
  – Maximum initial error is $||U0-U|| <= 2R_{max}/(1-\gamma)$
  – Require $\gamma^N * 2 R_{max} /(1-\gamma) <= \varepsilon$

function VALUE-ITERATION(*mdp*,ε) returns a utility function

inputs: *mdp*, an MDP with states *S*, transition model *T*, reward function R, discount γ

ε the maximum error allowed in the utility of any state

local variables U, U', vectors of utilities for states S, initially zero

δ , the maximum change in the utility of any state in an iteration

repeat

U<-U'; δ <- 0

for each state s **in** S **do**

Bellman update equation

if |U'[s] – U[s]| > δ **then** δ <- |U'[s] – U[s]|

until δ < ε(1- γ)/ γ

return U

- Implications
  - *N* grows rapidly as γ approaches 1 – affecting run-time
  - The smaller γ, the faster the convergence – gives agent short horizon
- Overall
  - MEU policy obtained by calculating U_i where i is # of iterations become optimal long before U_i has converged
  - No policy loss
  - Possible to get an optimal policy even when utility function estimate is inaccurate!

POLICY ITERATION

- Alternative way to find optimal policies
- Based on idea that if one action is clearly better than all others, then the exact magnitude of the utilities on the states involved do not need to be precise

- Beginning with some policy π_0 :
  - POLICY EVALUATION
    - given a policy π_i , calculate $U_i = U\pi$, the utility of each state if π were to be executed
  - POLICY IMPROVEMENT
    - calculate a new MEU policy π_{i+1} using one-step look-ahead based on U_i
- Terminate when policy improvement yields no change in the utilities
- Only finite # of policies for a finite space – must terminate – yields optimal policy

function POLICY-ITERATION(*mdp*) returns a policy
  inputs: *mdp*, an MDP with states *S*, transition modem *T*
  local variables: *U*, *U'*, vectors of utilities for states in *S*, initially 0
  π , a policy vector indexed by state, initially random

  repeat
    U <-POLICY-EVALUATION(π , *U*, *mdp*)
    unchanged <- true
    for each state *s* **in** *S* do
      if MEU > current **then**
        current <-MEU
  until *unchanged*?
  return π

- Easier to solve Bellman equations
  - Action in each state is fixed by the policy
  - At *i*th iteration, policy π_i specifies action in π_i (s) in state s
  - Simplifies Bellman equation relating utility of s to its neighbors
  - Produces LINEAR equations because the "max" operator is removed
  - Efficient in small state spaces
  - Solved in $O(n^3)$

- Modified Policy Iteration
  - Used in large state spaces
  - Not necessary to do EXACT policy evaluation
  - Perform some simplified iterative steps
  - More efficient
- Asynchronous Policy Iteration
  - Value and policy iteration require updating the utility of policy for all states at once
  - This only updates any subset of states
  - Given certain conditions on initial policy and utility function, guaranteed to converge to an optimal policy

Questions?