01DRO1
Decision Making under Uncertainty

2017/2018

Dynamic Programming. Solution of MDP. Value Iteration. Policy Iteration.

Lecture 4

13.3.2018

Readings:

- D.P. Bertsekas, *Dynamic Programming*, Prentice Hall, 1987, Sec 1.1 – 1.5
- Martin Puterman, *Markov decision processes*, John Wiley & Sons, 1994, Sec. 2.1-2.2

Reminder:

Mid-term assignment – Apr 3 (Lectures 1-6)

Homework topic approval due March 15.

Where are we?

Last time... Bayesian learning (BL)

Prediction in BL is a weighted average of the individual hypotheses predictions. Properties:

- no other prediction is correct more often than Bayesian (given prior), i.e. in a sense it is optimal
- *all* hypotheses are considered and weighted by their probability.

☹ for large hypotheses space BL can be intractable (sum over hypotheses)

Way out: approximate BL.

Last time.. Other than BL

Maximum A Posteriori (MAP) : prediction based on most probable hypothesis

☹ relies on one hypothesis => less accurate than BL

☹ finding the most probable hypothesis can be intractable and optimisation may be difficult

☺ prior can be used to penalise numerous hypotheses

Maximum likelihood (ML) prediction relies on one hypothesis only

☹ less accurate than BL and MAP predictions

☹ does not respect prior info

☹ easier to find than MAP

For increasing number of data *all* (MAP, ML and BL) predictions converge

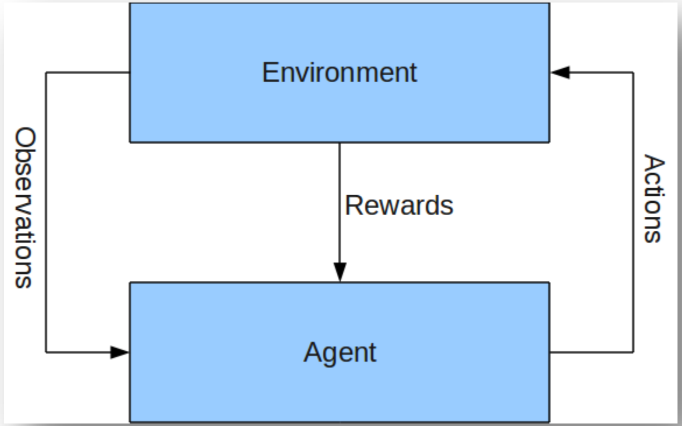
Today..

MDP: formalisation

MDP is defined by (S, A, T, r, h):

- S: finite set of all *states*, $|S| = n$
- A_s: finite set of *allowable actions* in state s.

  $A = \cup_{s \in S} A_s$, $|A| = m$
- h: *horizon* over which the agent will act
- $T: S \times S \times A \to [0,1]$, $T(s_{t+1}|s_t, a_t)$ – *state-transition function*
- $r: S \times A \times S \to \mathbb{R}$, $r(s_{t+1}, a_t, s_t)$ – *immediate reward function*



Goal find $\pi^{opt}: S \to A$ that maximises expected sum of immediate rewards, i.e.

$$\pi = \arg\max_{\pi} E\left[\sum_{t=0}^{h} r_t\left(s_{t+1}, a_t, s_t\right) \middle| \pi \right]$$

Assumptions

- *Markov dynamics* (history independence)

  $Pr(s_{t+1} | a_t, s_t, a_{t-1}, s_{t-1}, \ldots, s_0) = Pr(s_{t+1} | a_t, s_t) = T(s_{t+1} | a_t, s_t) = T(s_{t+1}, a_t, s_t)$
- *Markov reward*

  $Pr(r_t | a_t, s_t, a_{t-1}, s_{t-1}, \ldots, s_0) = Pr(r_t | a_t, s_t)$
- *Stationary MDP:* rewards and transition probabilities are independent of t:

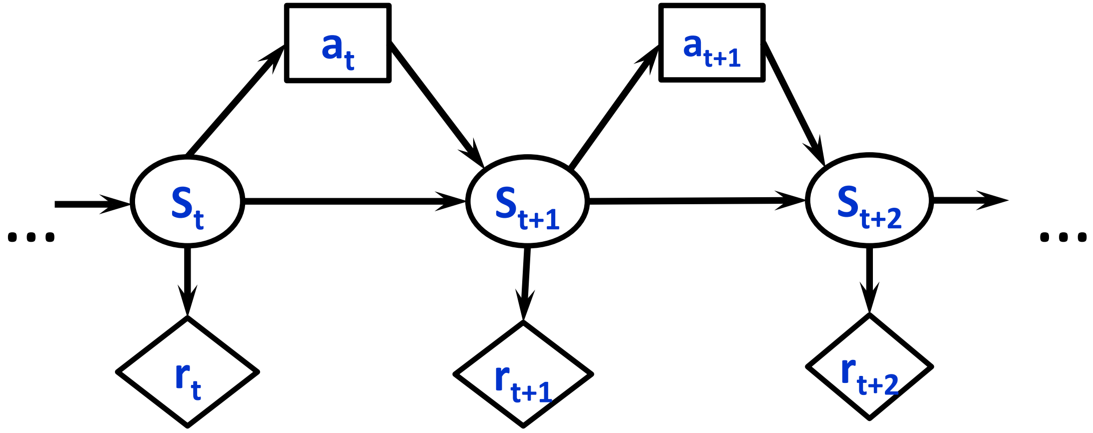
  $T(s_{t+1} | a_t, s_t) = T(s_{t'+1} | a_{t'}, s_{t'})$ *for all* t, t'
- *Full observability:*

  once action is performed, we can precisely observe the next state .

Reward function

As a result of choosing action $a \in A_s$ in state s at decision epoch t,

- the agent considers a reward that is associated with taking a particular action at state s.



Reward function

- we work with the reward that depends on the next state too $r(s_{t+1}, a_t, s_t)$ (not usual case). In DM when solution considers the expected value $E[r(s_{t+1}, a_t, s_t) | s_{t+1}]$, this reward can be reduced to the 'classical' one by defining:

$$r(s_t, a_t) = E\left[r(s_{t+1}, a_t, s_t) \middle| s_t = s, a_t = a\right] = \sum_{s_{t+1} \in S} T(s_{t+1} \mid a, s)\, r(s_{t+1}, a_t, s_t)$$

- Random reward: The reward may be a random variable whose distribution depends on (s_t, a_t, s_t). For DM purpose we can work with the expected value instead, i.e.

$$R(s_t, a_t) = E\left[r(s_{t+1}, a_t, s_t) \middle| s_t = s, a_t = a\right]$$

Utility function

Executing a policy π yields a sequence of rewards $r_1, r_2, \ldots$

How good is a policy π in a state s?

Define *utility function* $U(r_1, r_2, \ldots)$ to be some "quality measure" of a reward sequence

(The utility of a policy is the sum of the expected rewards on the path i.e utility is a random quantity).

- For *deterministic* actions criterion is sum of rewards obtained

  problem: infinite horizon => *infinite* result
- For *stochastic* actions, criterion is expected total reward obtained– again typically yields *infinite* value.

How do we compare policies of **infinite** value?

Utility function (cont.)

Utility “helps to understand” difference between sequences of states wrt the agent preferences. *Utility* function maps infinite *sequence of rewards* to a real number.

Considering stationary preferences

$$[s_0,s_1,s_2,\ldots] \succ [s_0,s_1',s_2',\ldots] \Leftrightarrow [s_1,s_2,\ldots] \succ [s_1',s_2',\ldots]$$

there are only two ways to combine immediate rewards over time:

- *Additive rewards*: $U([s_0,s_1,s_2,\ldots]) = \Sigma_t R(s_t)$
- *Discounted rewards*: $U([s_0,s_1,s_2,\ldots]) = \Sigma_t \gamma^t R(s_t)$, with discount factor $0 \leq \gamma \leq 1$

Utility of a state (a.k.a. value of state) is then defined:

$U(s)$ = *expected* (discounted) *sum of rewards* (until termination) *assuming best actions*

Discounting is the most analytically tractable approach.

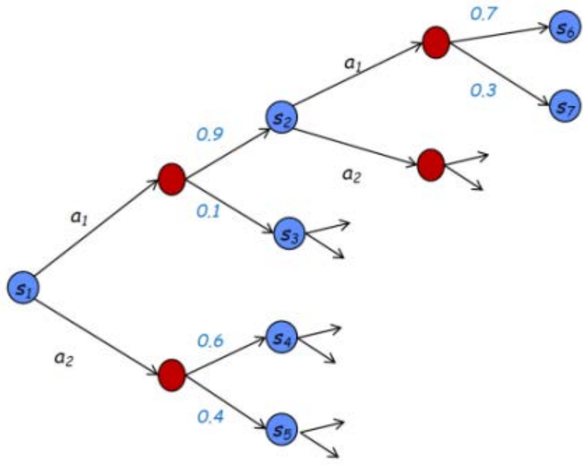
With a *proper* policy (with guaranteed terminal state) no discounting is needed.

Definition of utility of states leads to a simple relationship among utilities of neighboring states:

expected sum of rewards =

current reward + expected sum of rewards (after taking optimal action)

Value of a policy



Value function $V: S \to \mathbb{R}$

associates value with each state s (sometimes S x T)

$$V_\pi(s_t) = r_t(s_t, a_t) + E\,[V_{t+1}(s_{t+1}) \mid s_t, a]\}$$
$$= r_t(s_t, a_t) + \Sigma_{s \in S}\, T(s \mid s_t, a)\, V_{t+1}(s)],$$

$r_t(s_t, a_t)$ - immediate reward at t; π – policy

$E\,[V_{t+1}(s_{t+1}) \mid s_t, a]$ - expected reward in future periods $t+1,\ldots,h$.

$V_\pi(s)$ denotes value of policy π at state s

- total expected reward over horizon of interest starting in state s
- note that it is not immediate reward, $V_\pi(s) \neq r(s, a)$; $V_\pi(s)$ is expected utility and measures *utility of a policy. !!!*

see example: random paths=> different utilities. expectation of the utility is value

Value of a policy (cont.)

Value functions *partially* order the policies, but

- *at least* one optimal policy exists, and
- *all* optimal policies have the same value function

Quantities introduced

Action (control/decision) a_t - variable to be selected at epoch t.

Decision (control) rule $\pi_t : S \rightarrow A$ defines selection a_t in each state s_t, $a_t = \pi_t(s_t)$

DM policy (strategy) $\pi = \pi_1, \pi_2, \ldots,$ – a sequence of decision rules for all states

Reward (cost, loss) $r_t(s_{t+1}, s_t, a_t)$ immediate cost for choosing control a_t in state s_t

State transition function $T(s_{t+1} | s_t, a_t)$ action-dependent probability distribution

Expected reward (cost, loss) $E[r(s_{t+1}, a_t, s_t) | s_{t+1}]$

Value (cost-to-go) function $V_\pi(s_t)$ cumulative reward for starting at state s_t and acting according to π thereafter

π^{opt} and V^{opt} are optimal DM rule and corresponding optimal value function

Our objective: find optimal policy

Optimal policy π^{opt} maximises the expected utility (finite or infinite horizon)

$$U(s_0, s_1, s_2, \ldots) = \Sigma_{t=0,1,\ldots} \; \gamma^t R(s_t, a_t)$$

possibly subject to constraints on the environment performance.

$$V^{\text{opt}}_t(s_t) = max_{a \in A} \{ r_t(s_t, a_t) + \gamma \, E \, [V^{\text{opt}}_{t+1}(s_{t+1}) \,|\, s_t, a]\}$$
$$= max_{a \in A} \{ r_t(s_t, a_t) + \gamma \, \Sigma_{s \in S} \, T(s \,|\, s_t, a) \, V^{\text{opt}}(s)]\},$$
$$\pi^{opt}(\text{s}) = \text{arg}max_{a \in A} \{ r_t(s_t, a_t) + \gamma \, \Sigma_{s \in S} \, T(s \,|\, s_t, a) \, V^{\text{opt}}(s)]\},$$

Recursive properties of the value function – Bellman optimality equation (1957)

Dynamic programming

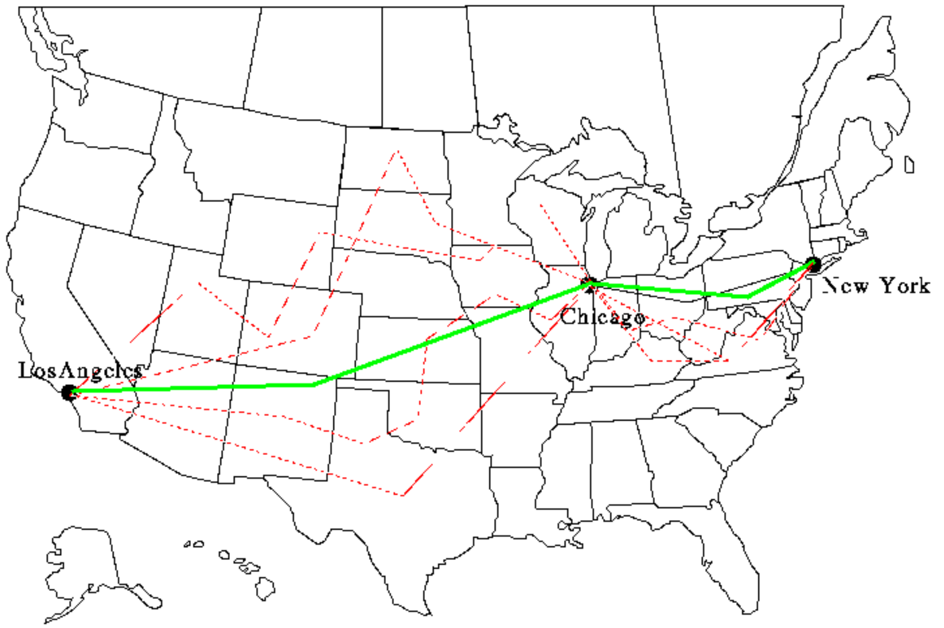
Dynamic programming is an optimization approach that transforms a complex problem into a sequence of simpler problems.

The principle of optimality: Any *optimal DM policy* has the property that, whatever the current state and decision, the remaining decisions must constitute an optimal policy with regard to the state resulting from the current decision.

Interpretation Bellman's optimality

- V – value function, cost-to-go
- V^{π}
- π
- V^{opt}
- π^{opt}

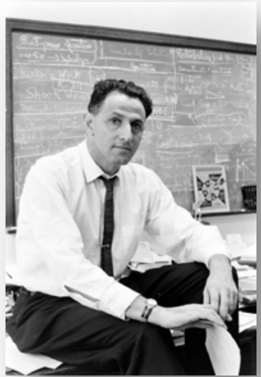
TRIVIAL EXAMPLE OF BELLMAN'S OPTIMALITY PRINCIPLE

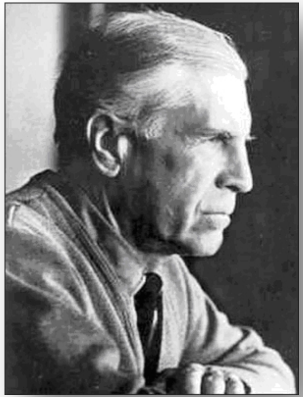


Bellman vs. Pontryagin

“In place of determining the optimal sequence of decisions from the fixed state of the system, we wish to determine the optimal decision to be made at any state of the system. Only if we know the latter, do we understand the intrinsic structure of the solution.”

—Richard Bellman, “Dynamic Programming”, 1957.





Pontryagin's maximum (or minimum) principle formulated in 1956 by Lev Pontryagin provides necessary conditions for optimality.

Bellman's equation is used to derive in different way (almost) the same equations that follow from Pontryagin's theory

MDP: solution approaches

Given (S, A, T, r, h)

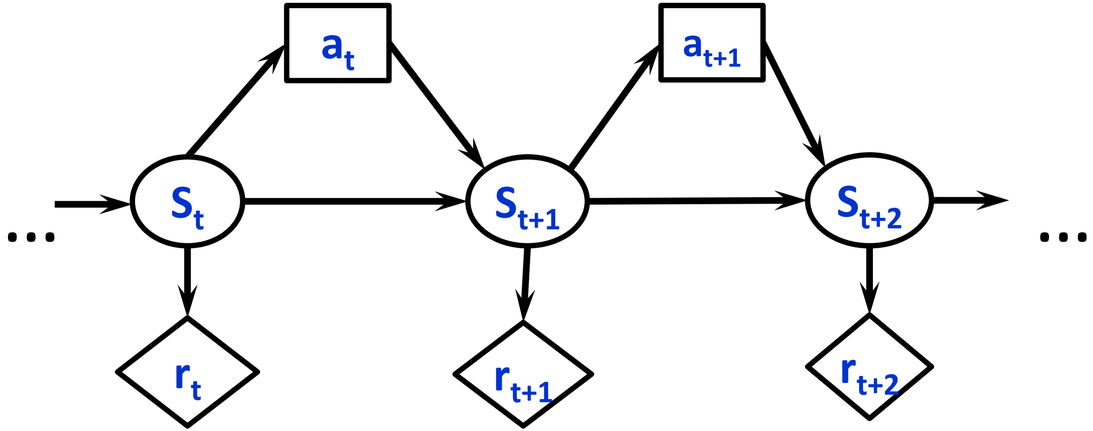
Goal to find $$\pi = \arg\max_{\pi} E\left[\sum_t r_t(s_{t+1}, a_t, s_t) | \pi\right]$$

- Finite Horizon Models
  - Dynamic Programming (Backward Induction)
- Infinite Horizon Models
  - Value Iteration
  - Policy Iteration (Modified Policy Iteration)
  - Linear Programming
  - Reinforcement Learning (Neuro-Dynamic Programming)

Due to simplicity discrete state-action spaces are considered.

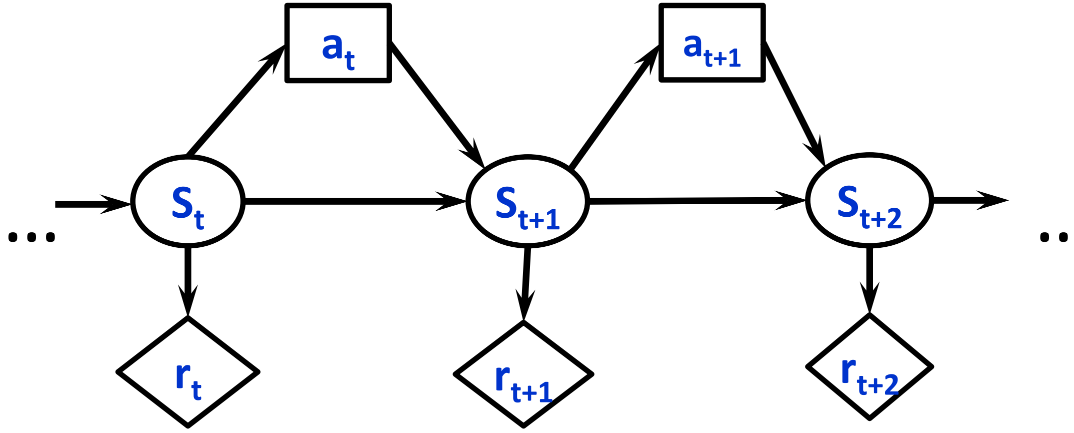
Value iteration, [Bellman 1957]

- Optimise decisions in *reverse* order
- Similar to variable elimination
- Performs dynamic programming

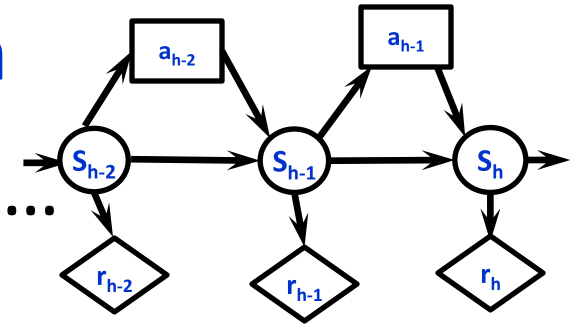


Value iteration (cont)

- Let h – horizon. At each $t=h,h-1,h-2,\dots,0$
- $a^{opt}_{t} = \text{argmax}_{a\in A} \{r_t(s_t,a_t) + \Sigma_{s\in S} T(s|s_t, a)\, V_{t+1}(s)]\},$



Backward Induction



- $t=h$ (terminal state) : $V_0(s_h)=r(s_h)$
- $t=h-1$ (one time step left):

  $V_1(s_{h-1}) = \max_{a(h)\in A} \{ r(s_{h-1}) + \Sigma_{s(h)\in S} \; T(s_h | s_{h-1}, a_{h-1}) \, V_o(s_h)\}$
- $t=h-2$ (two time steps left):

  $V_2(s_{h-2}) = \max_{a(h-2)\in A} \{ r(s_{h-2}) + \Sigma_{s(h-1)\in S} \; T(s_{h-1} | s_{h-2}, a_{h-2}) \, V_1(s_{h-1})\}$

…

$V_k(s_t) = \max_{a(t)\in A} \{r(s_t) + \Sigma_{s(t+1)\in S} \, T(s_{t+1} | s_t, a_t) \, V_{k-1}(s_{t+1})\}$ - Bellman equation

$a_t(s)= \text{argmax}_{a(t)\in A} \{r_t(s_t) + \Sigma_{s\in S} \, T(s_{t+1} | s_t, a_t) \, V_k(s_{t+1})]\}$,

Bellman eq. relates the value function to itself via the problem dynamics.

Value Iteration

- h iterations in total
- at each time step we need $|A|$ computation od n x n matrix times n vector
- slow to converge
- *Prioritized sweeping* is a more computationally efficient variation of value iteration. It decides what states to update first to maximise convergence. Rough idea: Puts all states in a priority queue in order of how much we think their values might change given a step of value iteration. Update states for which it is expected to be most significant based on the difference V_{k+1}- $V_{k.}$ Details see (Moore & Atkeson, 1993).

Value iteration: Algorithm

- Start with $V_0(s_{t+1}) = 0$ for all $s_{t+1} \in S$.
- for $k=1, \ldots, h$

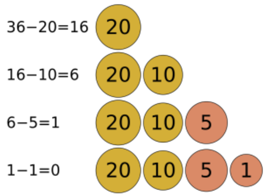
  Given V_k, calculate value update for all states $s_{t+1} \in S$:

$$V_{k+1}^{opt}(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + V_k^{opt}(s_{t+1})\right]$$

  $V_k(s_{t+1})$ is optimal k-stages to go value function = expected sum of rewards accumulated when starting from state s_{t+1} and acting optimally for a horizon of k steps;

- Given V^{opt}, use *greedy policy*:

$$\pi^{opt}(s_t) = \arg\max_{a_t \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + V^{opt}(s_{t+1})\right]$$



Notes

- Resulting policy is *optimal* $V_k^{\pi^{opt}}(s) \geq V_k^{\pi}(s)$ *for* $\forall\ \pi, s, k$
- optimal policy maximises value at each state
- optimal value function is unique, but optimal policy need *not* be unique
- Value equation for *fixed* policy value π

$$V^{\pi}(s_t) = \gamma \sum_{s_{t+1} \in S} T(s_{t+1}, \pi(s_t), s_t)\left[r(s_{t+1}, \pi(s_t), s_t) + V^{\pi}(s_{t+1})\right]$$

- Bellman equation for *optimal* value function V^{opt}

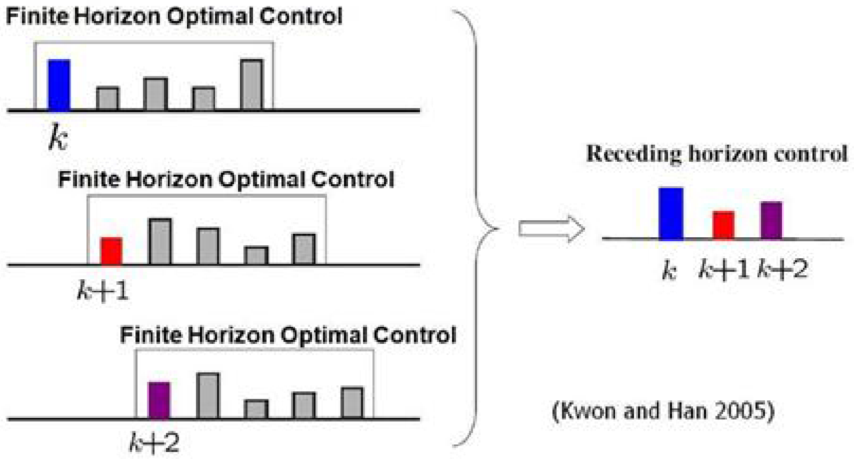
$$V^{opt}(s_t) = \gamma \max_{a_t \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + V^{opt}(s_{t+1})\right], \forall s_t \in S$$

Finite Horizon

For finite horizon (*h* is finite):

- optimal policy is *non-stationary*
- best action is different at each time step.

  Intuitive explanation: best action varies with the amount of time left
- Continuously executing *h-step optimal actions is known as* receding horizon control


Finite Horizon Optimal Control
k
Finite Horizon Optimal Control
$k+1$
Finite Horizon Optimal Control
$k+2$
Receding horizon control
k $k+1$ $k+2$
(Kwon and Han 2005)

Infinite Horizon

For infinite horizon (h is infinite):

- optimal policy is *stationary,* i.e. optimal action at a state s_t is the same action at all times. Note: no need to store argmax at each stage => efficient to store!

  Intuitive explanation: optimal action does not vary as the same, *infinite*, amount of time left at each time

Problem:

value iteration performs an infinite number of iterations.

Value iteration convergence

- Value iteration converges. At convergence, the optimal value function *V^{opt}* for the discounted infinite horizon problem satisfies the Bellman equations

$$V^{opt}(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + \gamma V^{opt}(s_{t+1})\right]$$

- When to stop value iteration? $\|V_{k+1} - V_k\| < \varepsilon, \forall s \in S$

$$\|V_{k+1} - V_k\| \le \gamma \|V_k - V_{k-1}\|$$

- this ensures $\max_{s \in S} \|V_{k+1} - V^{opt}\| < 2\varepsilon \frac{\gamma}{(1-\gamma)}$

$$\text{where } \|V\| = \max_s |V(s)| \text{ is a max-norm; } \varepsilon > 0$$

Infinite Horizon (cont.)

Problem:

value iteration performs an infinite number of iterations.

Solution:

Assuming a discount factor γ, rewards are scaled down by γ^k after k time steps. Thus rewards become insignificant for sufficiently large k, *as* $\gamma^k \rightarrow 0$.

Algorithm:

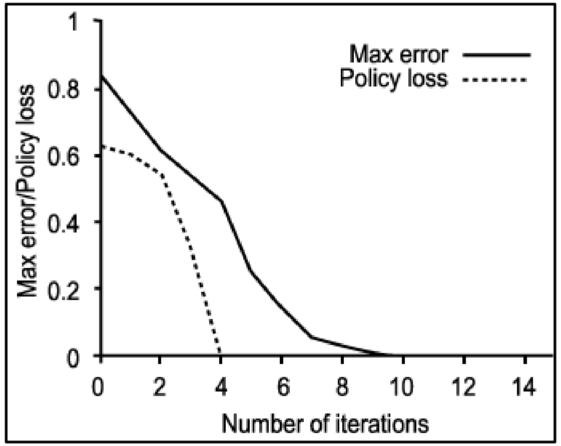
- select large enough k
- run value iteration for k steps
- apply policy found at the kth iteration

Policy loss

- Policy loss of π is the most the we can lose by executing π instead of the optimal policy π^{opt} $\left\| V^{\pi} - V^{opt} \right\|$

$$\left\| V_{k+1} - V_k \right\| \leq \varepsilon \Rightarrow \left\| V_k - V^{opt} \right\| \leq 2\varepsilon\gamma / (1-\gamma)$$

Note: often π becomes optimal long before V_k has converged => it is possible to get π^{opt} even if V_k is not accurate=> *policy iteration algorithm*



source: S.Russell, P.Norvig
Artificial Intelligence: A Modern Approach

MDP: solution approaches

Given (S, A, T, r, h)

Goal to find

$$\pi = \arg\max_{\pi} E\left[\sum_t r_t(s_{t+1}, a_t, s_t) \middle| \pi\right]$$

- Finite Horizon Models
  - Dynamic Programming (Backward Induction)
- Infinite Horizon Models
  - Value Iteration
  - **Policy Iteration** (Modified Policy Iteration)
  - Linear Programming
  - Reinforcement Learning (Neuro-Dynamic Programming)

Due to simplicity discrete state-action spaces are considered.

Policy iteration (PI)

PI optimises the policy *directly* instead of optimising the value function and then inducing a policy.

Policy iteration consist of *repetition of 2 steps*:

- *policy evaluation* – calculate value function for a fixed policy (not optimal value!) until convergence
- *policy improvement* – calculate a new policy for each *s* using one-step-ahead prediction with resulting converged (but not optimal) value function (from previous step) as future utilities

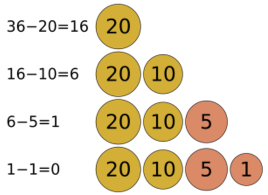
Algorithm continues until no improvement possible at any state.

Policy Iteration (Howard, 1960)

- PI computes V^{opt} and π^{opt} in a finite number of steps, typically on the same order as $|S|$.
- *Policy improvement is a* basic principle behind policy iteration
- Idea: Let π_k be a stationary policy, and V^{π_k} is its *value* function.

  A stationary policy π_{k+1} is called π_k - improving if it is a *greedy* policy with respect to V^{π_k} :

$$\pi_{k+1}(s_t) = \arg\max_{a_t \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + V^{\pi_k}(s_{t+1})\right], s \in S$$



Policy evaluation

Given: *fixed* current policy $\pi_{k.}$ Its value equals:

$$V^{\pi_k}(s_t) = \gamma \sum_{s_{t+1}\in S} T\big(s_{t+1}, \pi_k(s_t), s_t\big)\big[r\big(s_t, \pi_k(s_t), s_{t+1}\big) + V^{\pi_k}(s_{t+1})\big] \qquad (*)$$

(*) is a linear system of $|S|$ unknowns $\{V^{\pi_k}(s), s \in S\}$.

Way1: solve system of $n = |S|$ linear equations, T, $r(.)$ are constants. It requires $O(n^3)$ time; for small state space is the most efficient.

Way2: for large n => a number of simplified value iterations to get reasonable approximation of V^{π_k}; set $V_0^{\pi_k}(s_t) = 0$ and

$$V_{i+1}^{\pi_k}(s_t) = \max_{a\in A} \sum_{s_{t+1}\in S} T\big(s_{t+1}, \pi_k(s_t), s_t\big)\big[r\big(s_t, \pi_k(s_t), s_{t+1}\big) + V_i^{\pi_k}(s_{t+1})\big]$$

Policy iteration: algorithm

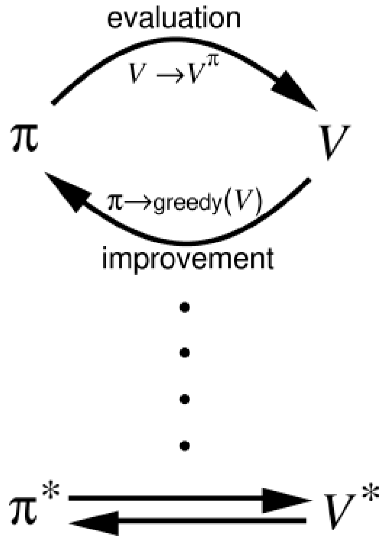
- Initialisation: choose a random stationary policy π_0
- Loop: for k=0,1,2,…
  - Policy evaluation: compute V^{π_k} either via solving linear system (Way1) or via Bellman updates (Way2)
  - Policy improvement: compute new policy π_{k+1} as a greedy policy wrt fixed V^{π_k} :

$$\pi_{k+1}(s_t) = \arg\max_{a_t \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + V^{\pi_k}(s_{t+1})\right], \quad \forall s_t \in S$$

  - Replace π_k with π_{k+1}
  - Stop if $V^{\pi_{k+1}} = V^{\pi_k}$ i.e. until no improving action is possible in any state

Remarks on PI

- Convergence: assured
- Flexibility: PI needs to improve policy at one state only (not each state)
- Provides exact value of optimal policy
- PI generally converges much faster than VI



Value (VI) vs. Policy iteration (PI)

- Since every step leads to improvement and there are finitely many policies, **PI** terminates at optimal solution and has finite-time convergence.
- Computational complexity:

VI requires $O(|A|\cdot|S|^2)$ per step, but number of steps can be large, especially when $\gamma \to 1$. Usually $|A| \ll |S|$ **=>** variants of VI

PI:

- requires solving large linear systems, $O(|A|\cdot|S|^2+|S|^3)$ operations per step but fewer number of steps.
- solving the linear system for policy evaluation may be time consuming (e.g., for large state spaces) **=>** variants of PI

Some Variants on Value Iteration (no exam)

- Standard VI: $V_k(s)$ is held fixed while all entries $V_{k+1}(s)$ updated.
- Gauss Seidel VI updates each element of $V_k(s) = V_{k+1}(s)$ once the latter is computed (i.e. *before* proceeding to the next state), and continue the calculation with the new value.
  - assumes some ordering of states
  - speeds up convergence
  - important: $V_k(s)$ is *no* longer k-stage-to-go value function
- Asynchronous VI: at each iteration *a subset* of all states is updated $V_k = V_{k+1}$
  - if each state is updated infinitely often, $V_k(s)$ converges to $V^{opt}(s)$
  - can be used to focus the computational effort on "important" parts of a large state-space and for on-line algorithms (RL). Selection a subset.

Modified (aka Generalized/Optimistic) **Policy Iteration** (no exam)

- Combines policy improvement steps with VI for policy evaluation => *avoid* computing exact policy evaluation
- Algorithm: set initial value of V_0, then perform iteratively:
  - greedy *policy* computation: compute π_k, a greedy policy wrt V_k
  - *partial* VI: perform m_k steps of value iteration V_k ->V_{k+1}
- Notes:
  - to evaluate policy it uses several iterations of successive approximations instead of solving linear system
  - guarantees convergence of V_k to V^{opt}. In practice a few iterations are enough to get estimate V_{k+1} ensuring improved π_{k+1}
  - important practical task: selection of m_k. Extreme values of m_k reduces the algorithm to standard VI or PI

MDP: solution approaches

Given (S, A, T, r, h)

Goal to find
$$\pi = \arg\max_{\pi} E\left[\sum_t r_t(s_{t+1}, a_t, s_t) \middle| \pi \right]$$

- Finite Horizon Models
  - Dynamic Programming (Backward Induction)
- Infinite Horizon Models
  - Value Iteration
  - Policy Iteration (Modified Policy Iteration)
  - **Linear Programming**
  - Reinforcement Learning (Neuro-Dynamic Programming) – Lecture 6

Due to simplicity discrete state-action spaces are considered.

Linear Programming (LP)

- Recall $V^{opt}(s_t) = \max_{a \in A} \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + \gamma V^{opt}(s_{t+1})\right], \forall s_t \in S$

- Optimal DM problem is formulated as linear programming, which can be solved efficiently using standard LP solvers.
- LP formulation: to find optimal V^{opt}

$$\min_V \sum_{s_{t+1} \in S} V(s_{t+1}), \text{ such that } \forall a_t \in A, \forall s_t \in S$$

$$V(s_t) \geq \sum_{s_{t+1} \in S} T(s_{t+1}, a_t, s_t)\left[r(s_{t+1}, a_t, s_t) + \gamma V^{opt}(s_{t+1})\right],$$

- Number of inequality constraints is |S|x|A|

Some Variants on the Basic MDP Model

- There may be a *continuum of states* and/or actions
- Decisions may be made in *continuous time*
- Rewards and transition rates may *change* over time
- Environment state may be *not observable*
- Some model parameters may not be known

Final remarks:

Easy and elegant formulation, but how to select all components?

- definition of state space *S* may require:

  structure estimation; hypothesis testing
- selection of transition functions:

  knowledge elicitation; learning
- reward function:

  preferences elicitation; knowledge elicitation
- initial state:

  is randomly chosen according to a given probability => need to be specified

  knowledge elicitation; learning
- value function computation => *approximations* (basis of RL)
- other computational issues => …

Limitations of MDPs

- Curse of dimensionality
  - As the state and/or action space become larger, it becomes computationally very difficult to solve the MDPs
  - There are more memory-efficient methods than Policy Iteration and Value Iteration
  - There are also some solution techniques that find near-optimal solutions in a short time
- Data requirements
  - For each action and state pair, we need a transition probability matrix and a reward function

MDP: Approximate Solution Techniques

- Finding the optimal policy is polynomial time in the state space size
- Examples of approximate solution techniques
  - State aggregation and action elimination
  - AI techniques (reinforcement learning)
  - Approximate linear programming (van Roy)
  - a very active area in the last decade

Partially Observable MDP

next time