SIT796 Reinforcement Learning

MDPs and Dynamic Programming

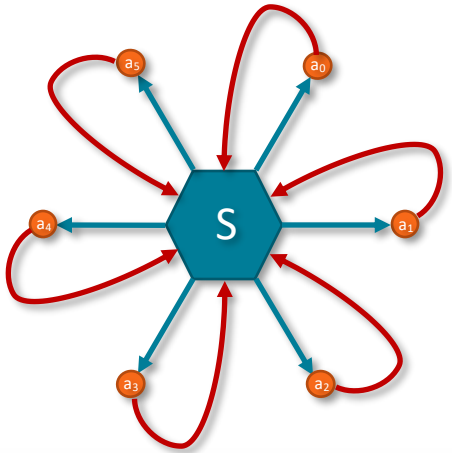
DEAKIN UNIVERSITY

Presented by:
Dr. Thommen George Karimpanal
School of Information Technology

Week 2 Recap:

The Problem

- Each step you are faced with the same choice of actions.
- After choosing an action you end up in the same state but will have received a reward for the action taken
  - Positive, negative or zero
  - The reward is stochastic – we will assume a normal distribution, but any distributions can be studied
- Your objective is to maximise your reward over a period of time.
  - Can often focus on minimising ***regret*** instead.



Examples

- Slot machines – people often search for the 'luckiest' machine
- Doctor finding the best medicine for your medical condition
- Recommendation systems.
- Anomaly detection
- …

https://www.rubyfortune.com/blogrfc/images/uploads/2019/07/Pixabay-Vintage-Slots.png

So, which action should you choose?

Week 2 recap: Basic MAB Algorithm

Initialise: for $a = 1$ to k

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \underset{a}{\text{argmax}}\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \underset{a}{\text{random}}\, Q(a) & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow \mathrm{N(A)} + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}[R - Q(A)]$

The above algorithm has five primary steps that are often replaced with different approaches.

- Initialisation (prior to loop)
- Action selection
- Gather any reward (in full problem this will include getting the new state)
- Update step counter (or other internal parameter adjustments)
- Update Q estimate (update new state to current state)

Optimistic initialisation

Ontrack task: Value Calculation

a) In Fig 1, what are the discounted and undiscounted values $V_\pi(s)$ and $V_{\pi^*}(s)$?

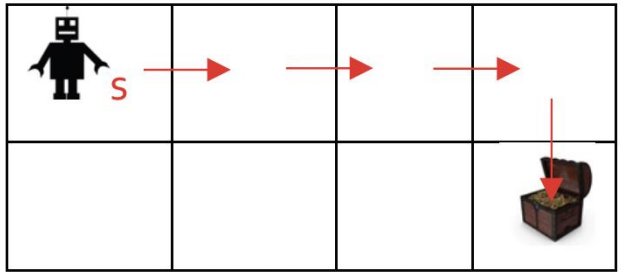


Fig 1.

The red arrows represent the policy π

Is the shown policy the optimal one π^*?

b) In Fig 2, what are the discounted and undiscounted values $V_\pi(s)$ and $V_{\pi^*}(s)$?

Discounted; set $\gamma = 0.9$
Undiscounted; set $\gamma = 1$

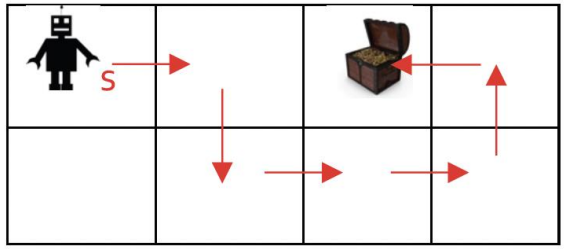
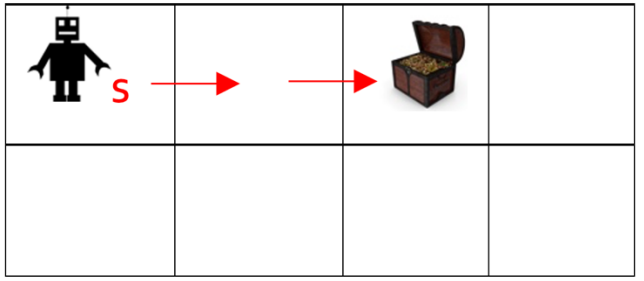


Fig 2.

What is the optimal policy π^* here?



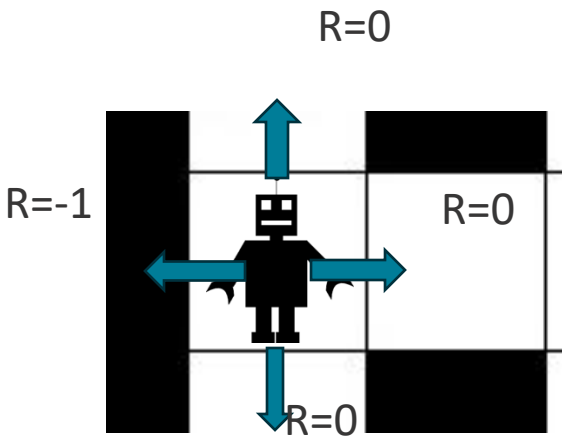
Questions from workshop:

How to deal with continuous states and continuous actions?

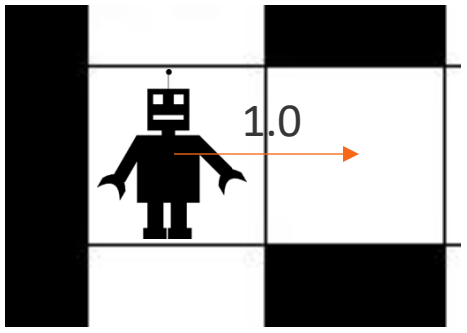
The Problem

Formally, the RL problem is formulated as a Markov Decision Process (MDP)

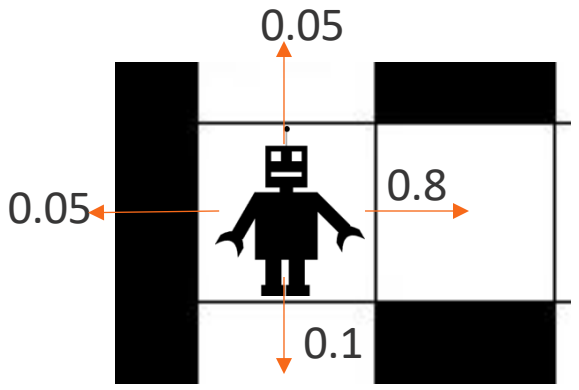
- An MDP is a tuple $M=\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \gamma, \mathcal{R}\}$
  - $\mathcal{S}$ - The set of possible states
  - $\mathcal{A}$ - The set of actions the agent can take
  - $\mathcal{T}$ - transition probabilities
  - γ - The discount rate or the discount factor.
  - $\mathcal{R}$ - A reward distribution function conditioned.



Reward function



Deterministic transition



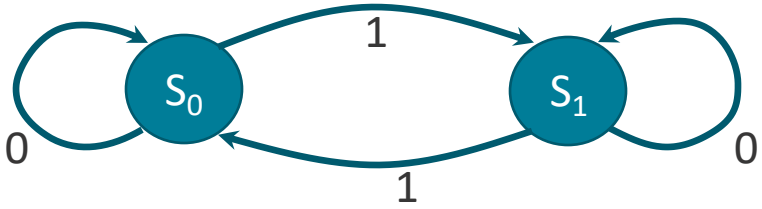
Probabilistic transition

Finite State Machine

Consider: When you press a button on your Spotify player to skip forward one song it combines the input you just entered with preceding events.

- Such as the selection of shuffle or repeat
- Current playlist selected
- Other music that matches your taste

Thus, a machine is a system that accepts input values, possibly produces outputs, and has an internal mechanism to track previous inputs/events



A Finite state machine is a graph representation

- A finite, directed, connected graph
- Contains a set of states (graph nodes)
- A set of inputs/actions (arcs)
- A state transition function, describing the effect of the inputs on the states

	0	1
S_0	S_0	S_1
S_1	S_1	S_0

Probabilistic Finite State Machine

A PFSM is a FSM where the next state function is a probability distribution over the full set of states of the machine.

- May contain a number of transitions for each input
- Each transition has a probability (0-1)
- The transitions for an input should add to 1
- The probability represents the random chance of selecting that path

Example – Recycling Robot

Background:

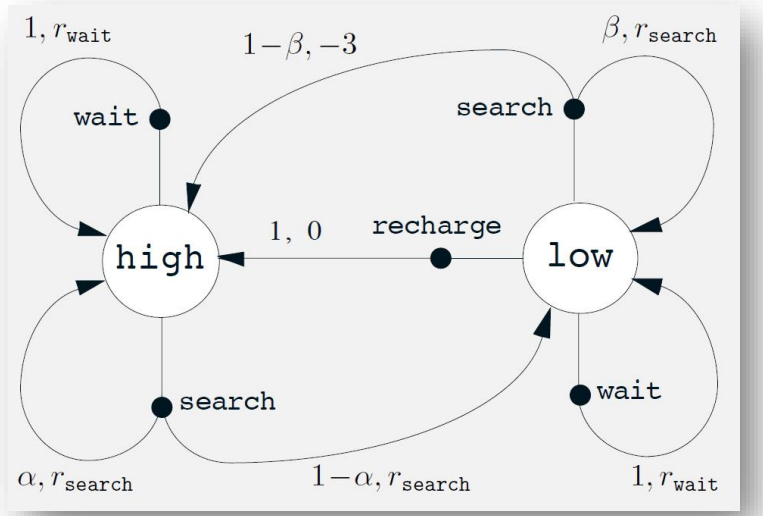
- Robot must try and collect rubbish for recycling without running out of power.

Problem

- Simplified problem assumes low level control is handled by other systems
- Learn to collect rubbish only when the risk of running out of power is low.

Defining an MDP – episodic

- State – Symbolic discrete states
  - Currently has ‘high’ or ‘low’ charge.
- Actions – Produces a vector of actions (not a single action)
  - Wait – no chance of changing state, but smaller chance of getting a reward.
  - Search for rubbish, when state is ‘high’ there is a $1-\alpha$ chance it will change to state ‘low’.
  - Search for rubbish, when state is ‘low’ there is a $1-\beta$ chance battery will go flat.
  - Recharge (only from state ‘low’) – transitions to state ‘high’, no reward
- Rewards (positive and negative rewards)
  - Small chance of finding a can when searching or waiting. Each can collected is a +1 reward.
  - A reward (penalty) of -3 if the robot needs rescuing as it ran out of power.



s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-

Sutton and Barto, Page 74

https://www.environmentalleader.com/wp-content/uploads/2017/03/robot.jpg

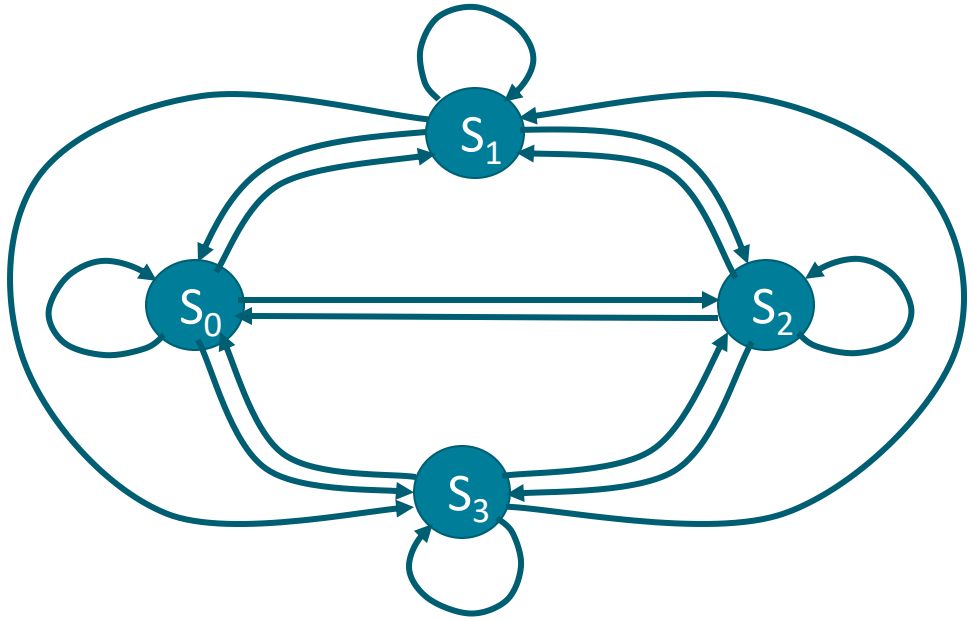
Markov Model

Named after Andrey Markov, a Markov Model is a *stochastic* model used to model randomly changing systems.

The same as a probabilistic finite state machine except it ignores the input values.

- In other words each transition is only labelled with a probability

Can be used to model a large number of natural phenomena and processes



First-Order Markov Model (Markov Chain)

An Observable Markov Model is *first-order* if the probability of it being in the present state S_t at any time t is a function only of its being in the previous state S_{t-1} at the time $t-1$, where S_t and S_{t-1} belong to the set of observable states $\mathcal{S}$.

- For all states S_i and S_j we can say $p_{ij} = P(S_{t+1} = S_j | S_t = S_i)$ for all time, t. These probabilities are assumed to be stationary.
- These probabilities create a transition probability matrix, eg matrix M below.
- Note, there are also second, third-order, … Markov models that consider multiple nodes contribute to the transition probability.
  - While there is research into second-order approaches to RL – we will not consider them

As a First-Order Markov Model moves from one discrete node to the next based only on the previous node it is also referred to as a Markov Chain

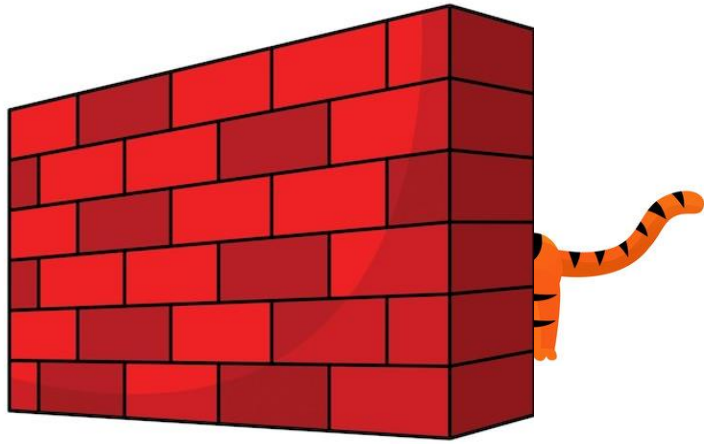
	S_0	S_1	S_2	S_3	Total
S_0	0.4	0.3	0.2	0.1	1.0
S_1	0.2	0.3	0.2	0.3	1.0
S_2	0.1	0.3	0.3	0.3	1.0
S_3	0.2	0.3	0.3	0.2	1.0

Partially Observable MDPs (POMDPs)

What if we can’t know the state completely?

Eg: We see what looks like a tiger’s tail, but the rest of the tiger is behind a wall

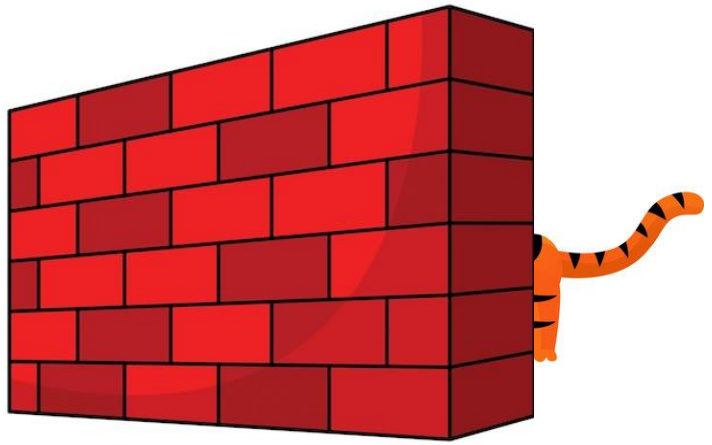
Can we say for sure that the state should contain `Tiger’?



Partially Observable MDPs (POMDPs)

A *POMDP* is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{P}, \mathcal{R}, \mathcal{Z}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{A}$ is a finite set of actions
- $\mathcal{O}$ is a finite set of observations
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
- $\mathcal{Z}$ is an observation function, $\mathcal{Z}^a_{s'o} = \mathbb{P}[O_{t+1} = o \mid S_{t+1} = s', A_t = a]$
- γ is a discount factor $\gamma \in [0, 1]$.



Source: https://www.davidsilver.uk/wp-content/uploads/2020/03/MDP.pdf

Multi-Objective MDP (MOMDP)

An MDP provides a single scalar reward each time step indicating the success/failure of the agent.

- The objective of the agent is to maximise this reward – hence achieving its goal

However, many problems naturally have multiple *conflicting* objectives.

- Shoot down the enemy plane without being shot down yourself
- Release enough water to power the city but save some for future droughts
- Make as much profit as possible while release as little green house gases as possible

A Multi-objective MDP (MOMDP) is an MDP except instead of a single reward

- Has a vector of rewards – one for each objective

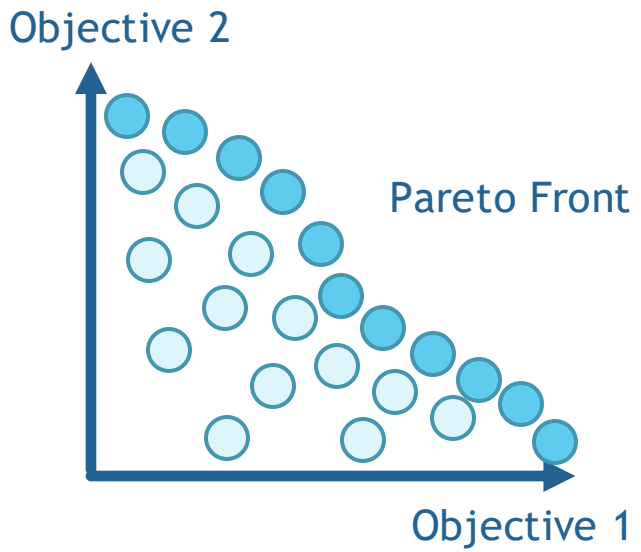
Instead of finding a single optimal policy it works on a set of pareto optimal policies

Two main types of MORL Problems:

Single-policy MORL : aims to find a single policy on the front which is a good match to some pre-defined specifications

Multi-policy MORL aims to find a good approximation to this front

- Solutions accurate (close to the actual front)
- Solutions are well distributed along the front
- Similar extent to the actual front



https://www.researchgate.net/profile/Gary_Yen/publication/3949961/figure/fig2/AS:830625074384897@1575047858467/Architecture-of-a-simple-hierarchical-RL-agent.ppm

Markov Model Cheat Sheet

Markov Models		Do we have control over the state transitions?	
		No	Yes
Are the states completely observable?	Yes	Markov Chain	MDP (Markov Decision Process)
	No	HMM (Hidden Markov model)	POMDP (Partially observable MDP)

Source: https://www.cs.cmu.edu/~ggordon/780-fall07/lectures/POMDP_lecture.pdf

MDP Types

Basis		
Horizon:	Finite	Infinite
State Transitions:	Deterministic	Stochastic
Terminal condition:	Episodic	Continuous
Discounting:	Discounted (γ<1)	Undiscounted (γ=1)
Observability:	Fully observable	Partially observable

SIT796 Reinforcement Learning

Dynamic Programming

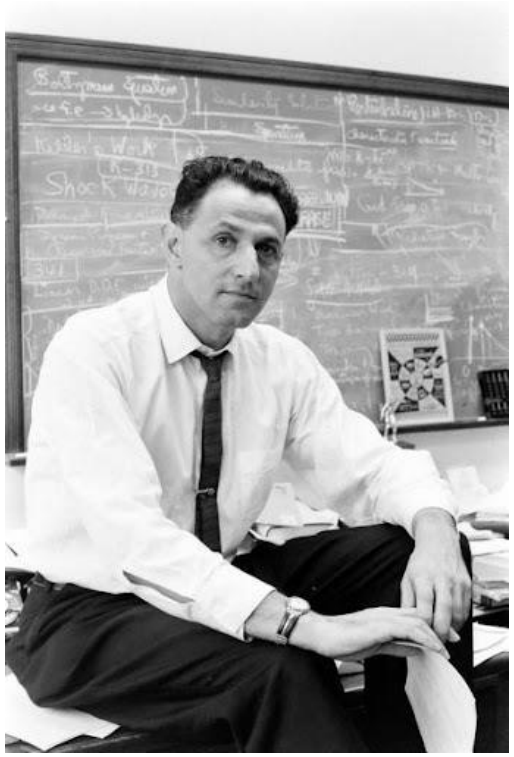
DEAKIN UNIVERSITY

Presented by:
Dr. Thommen George Karimpanal
School of Information Technology

Dynamic Programming

Developed by Richard Bellman (1950s)

"An interesting question is, 'Where did the name, dynamic programming, come from?' The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research. I'm not using the term lightly; I'm using it precisely. His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical. The RAND Corporation was employed by the Air Force, and the Air Force had Wilson as its boss, essentially. Hence, I felt I had to do something to shield Wilson and the Air Force from the fact that I was really doing mathematics inside the RAND Corporation. What title, what name, could I choose? In the first place I was interested in planning, in decision making, in thinking. But planning, is not a good word for various reasons. I decided therefore to use the word, 'programming.' I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying—I thought, let's kill two birds with one stone. Let's take a word that has an absolutely precise meaning, namely dynamic, in the classical physical sense. It also has a very interesting property as an adjective, and that is it's impossible to use the word, dynamic, in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name. It was something not even a Congressman could object to. So I used it as an umbrella for my activities" (p. 159).



Source http://www.breves-de-maths.fr/richard-bellman-et-la-programmation-dynamique/

Source: https://pubsonline.informs.org/doi/pdf/10.1287/opre.50.1.48.17791

Dynamic Programming

What is Dynamic Programming?

Mathematical optimisation technique used to find optimal solutions to MDPs

It requires the availability of the perfect model of the system (i.e., full knowledge of the transition probabilities)

DP can be used to compute optimal value functions using `Bellman update equations'

Dynamic Programming

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big],$$

Consequences of current action

The best it can do in the future

$$q_*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$
$$= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big],$$

"The only thing that matters is where you are (state) and what you do (actions)"

Policy Evaluation (Prediction)

We want to evaluate a given policy π

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big]
\end{aligned}
$$

$$
G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}
$$

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma \big(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots \big) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

Iteratively applying this update will result in the value function

Policy Evaluation (Prediction)

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$ arbitrarily, for $s \in \mathcal{S}$, and $V(terminal)$ to 0

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

$$\Delta = \max_{s \in \mathcal{S}} \left| v_{k+1}(s) - v_k(s) \right|$$

Policy Improvement

Now we know how good it is to follow the current policy from state s $v_\pi(s)$

Would it be better to change to a new policy from s?

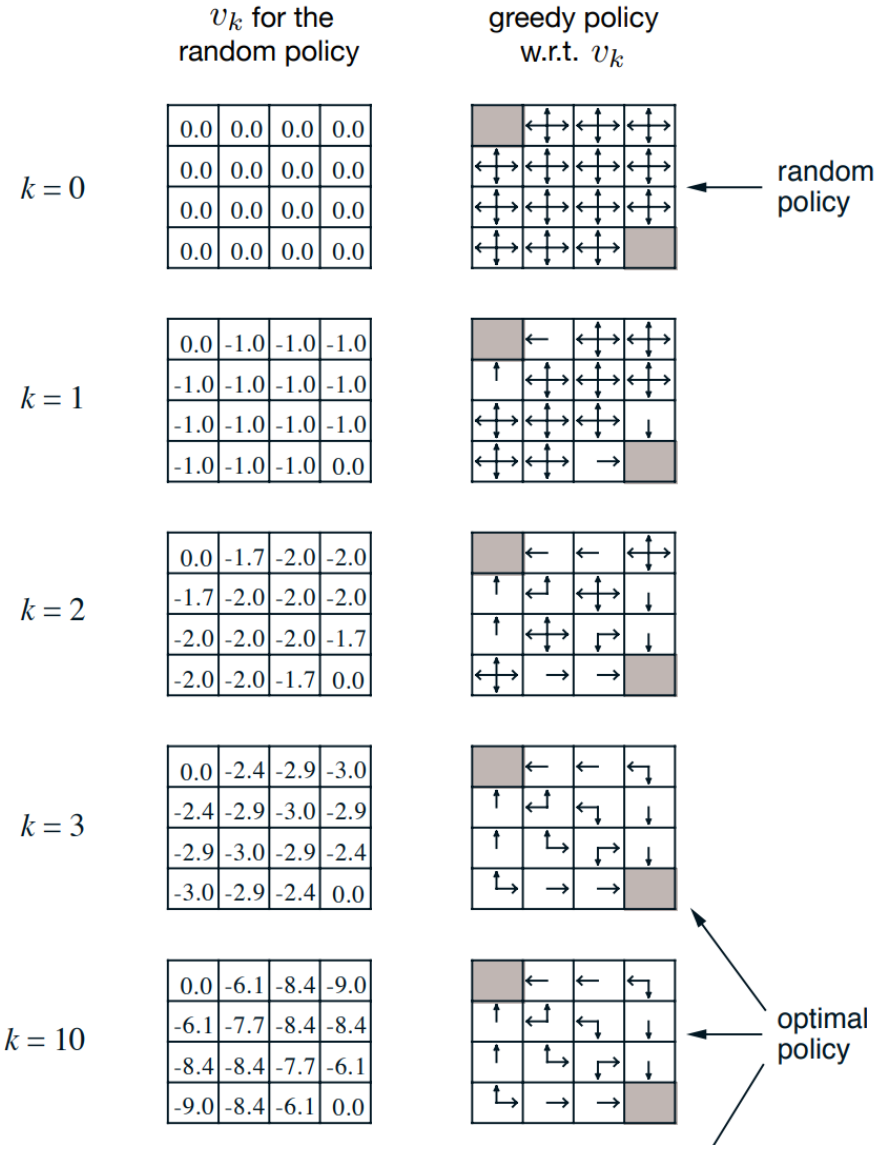
One way to find out: Choose $a \neq \pi(s)$ and then continue following the original policy

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_\pi(s')\Big]. \end{aligned}$$

The new policy π' is better if: $q_\pi(s, \pi'(s)) \geq v_\pi(s)$

Comes from the Policy Improvement Theorem

Policy Improvement (Example)



Policy Iteration

Evaluate policy, then improve, then evaluate again and then improve again …… till convergence!

Disadvantage: Requires policy evaluation loop

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\textit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \left[r + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   $\textit{policy-stable} \leftarrow \textit{true}$
   For each $s \in \mathcal{S}$:
   $\textit{old-action} \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma V(s') \right]$
   If $\textit{old-action} \neq \pi(s)$, then $\textit{policy-stable} \leftarrow \textit{false}$
   If $\textit{policy-stable}$, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value Iteration

The Bellman optimality eq. is turned into an update rule.

Evaluates and improves policy simultaneously

Converges faster than policy iteration.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

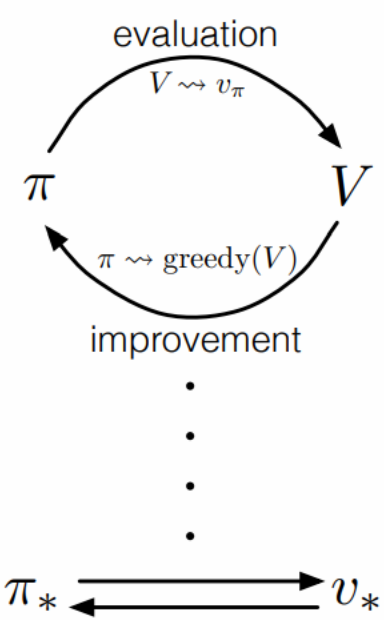
Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

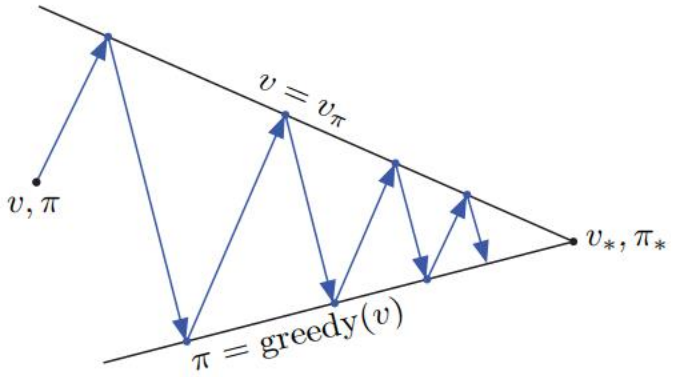
Generalized Policy Iteration

When does evaluation stabilise?

When does improvement stabilise?

If both stabilise, then the value function and policy must have converged (optimum)





DP disadvantages?

Not suitable for very large problems

No. of states grows exponentially with no. of state variables – *Curse of Dimensionality!*

Requires sweep over entire state space – Asynchronous DP is a solution

Prioritized sweeping: Use $|v - V(s)|$ as a basis for deciding which states to update

It requires a model of the system – RL does not need a model in general

Bootstrapping – updating an estimate based on another estimate -errors can add up!

Readings

This lecture focused on the MDPs, Finite state machines and Dynamic programming.

- Ensure you understand what was discussed here before moving to the subsequent topics

For more detailed information see Sutton and Barto (2018) Reinforcement Learning: An Introduction

- *Chapter 3: Finite Markov Decision Processes*
- http://incompleteideas.net/book/RLbook2020.pdf