Applied Machine Learning: Examples in the ICT domain

Prof. Andrea Zanella

zanella@dei.unipd.it

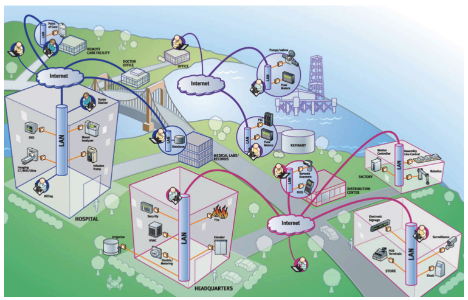
- office: +39 049 8277770
  fax : +39 049 8277699
- email: zanella@dei.unipd.it
- web : http://www.dei.unipd.it/~zanella

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

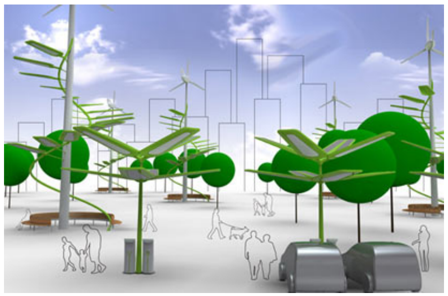
SIGNET

SIGNET people

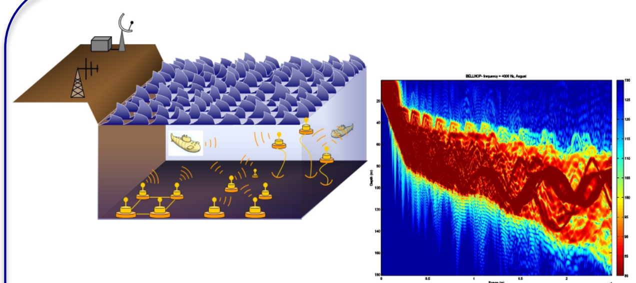
Main research areas…



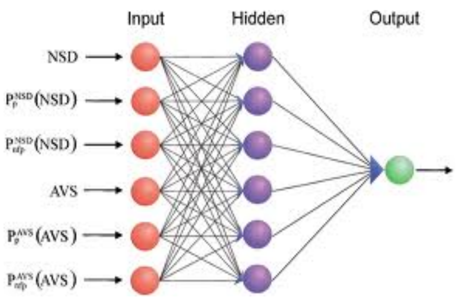
Next generation mobile & IoT



Energy harvesting

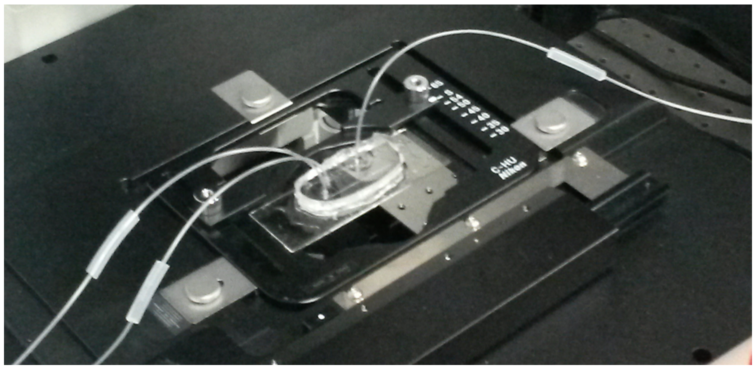


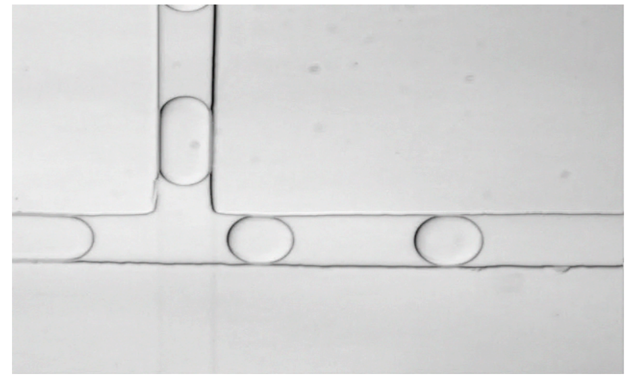
Underwater communications

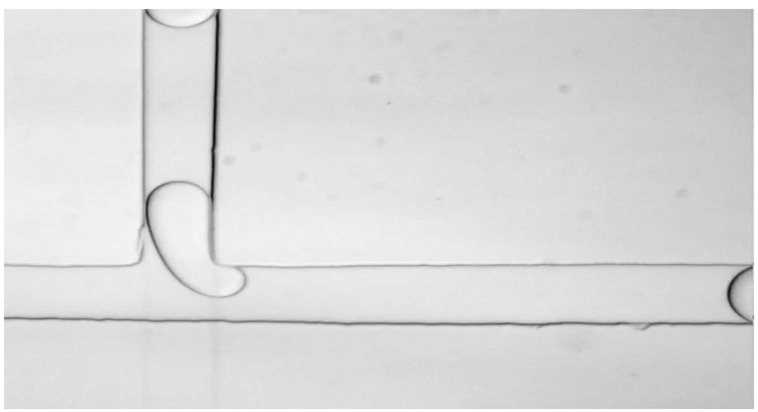


Human data analytics

and some more exotic stuff…







Outline (tentative)

- Introduction to reinforcement learning
- Deep Q-learning for mobile multimedia streaming applications
- MultiArmed bandit for HetNet configuration
- Other examples of ML applications to ICT
- Conclusions

What does "learning" actually mean for machines?

Machine "learning"

"A computer program is said to learn from **experience** E with respect to some class of **tasks** T and **performance measure** P, if its performance at tasks in T, as measured by P, improves with experience E"*

* Mitchell, T. M. (1997). Machine Learning. McGraw-Hill, New York.

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Human Learning
vs
Machine Learning

How do (small) humans learn?

Imitative Learning

- Small kids (toddlers) first learn by imitation
  - “The most striking findings were that toddlers were able to learn a new action from observing completely unfamiliar strangers who did not address them and were far less likely to imitate an unfamiliar model who directly interacted with them.” [1]

[1] Priya M. Shimpi, Nameera Akhtar, Chris Moore,”Toddlers’ imitative learning in interactive and observational contexts: The role of age and familiarity of the model,” Journal of Experimental Child Psychology, Volume 116, Issue 2, 2013, https://doi.org/10.1016/j.jecp.2013.06.008

Is there anything similar in ML?

Imitation learning in machines

- Manual training of industry robots
  - Robotic arms can be manually moved by an operator to learn how to perform a repetitive task, which they then replicate autonomously
  - More a new form of programming than actual machine "learning"

Transfer learning

- Transfer Learning is closer to our idea of Imitative Learning
  - It consists in transferring knowledge gained by an ML algorithm while solving one problem to a different but related problem
    - Eg, knowledge learned by an algo that detects cars in pictures can be transferred to an algo that recognizes trucks
  - Makes it possible to greatly speed up learning of other ML algorithms in new problems

Is that sufficient?

Learning by experience

Of course not!

- Children also need to make their own experience in order to learn

- Experience learning is based on
  - Exploring (e.g., by playing)
  - Experimenting (by trial and errors)
  - Asking questions (to cut ties)

Exploring through "play"

Picture taken from: https://www.whitbyschool.org/passionforlearning/how-do-children-learn-through-play

Is there anything similar in ML?

Play → Pre-training

- The ML equivalent of children plays could be the **pre-training** of ML algorithm by using **hyper-simplified models** of the target problem
- This practice is particularly beneficial for ML algorithms that require massive datasets for training

The theory theory

- **Cognitive development is like theory revision in science** [2]

- Children construct intuitive theories of the world and alter and revise them as the result of new evidence

[2] Gopnik A, Wellman HM. Reconstructing constructivism: causal models, Bayesian learning mechanisms, and the theory theory. *Psychol Bull*. 2012;138(6):1085–1108. doi:10.1037/a0028044

Statistical information

- Children gradually change the probability of multiple hypotheses rather than simply rejecting or accepting a single hypothesis

- Evidence leads children to gradually revise their initial hypotheses and slowly replace them with more probable hypotheses

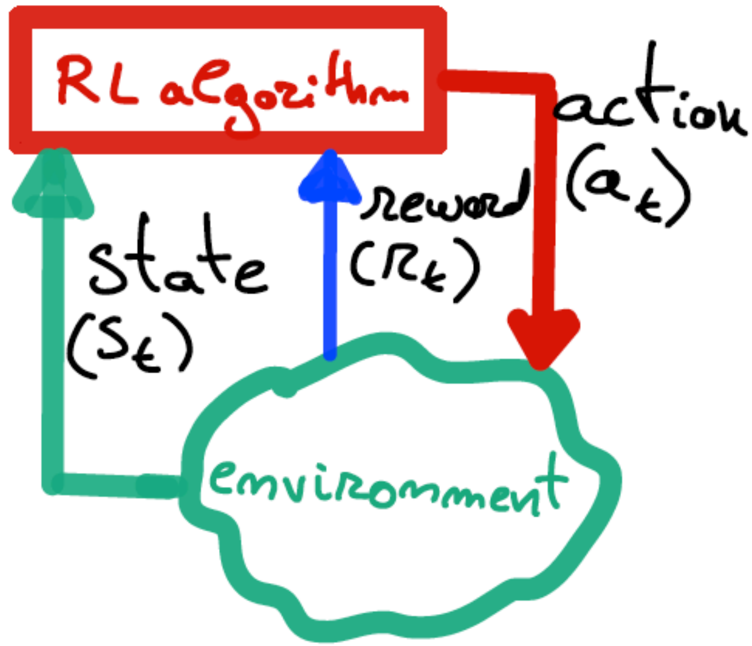
Experimenting (trial & errors)

- A simple (?) experimental test…

Is there anything similar in ML?

Sure: Reinforcement Learning!

Reinforcement Learning



Problems involving an **agent** interacting with an **environment**, which provides numeric **reward** signals

Goal: Learn how to take actions in order to maximize reward

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn a *function* to map x → y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Cat

Classification

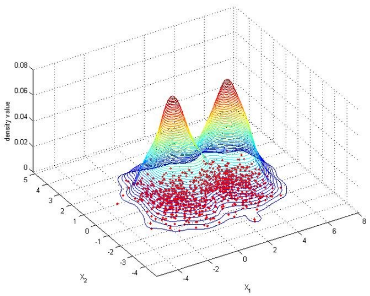
Unsupervised Learning

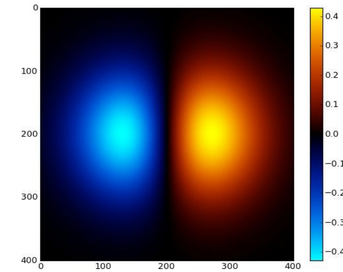
Data: x
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

1-d density estimation





2-d density estimation

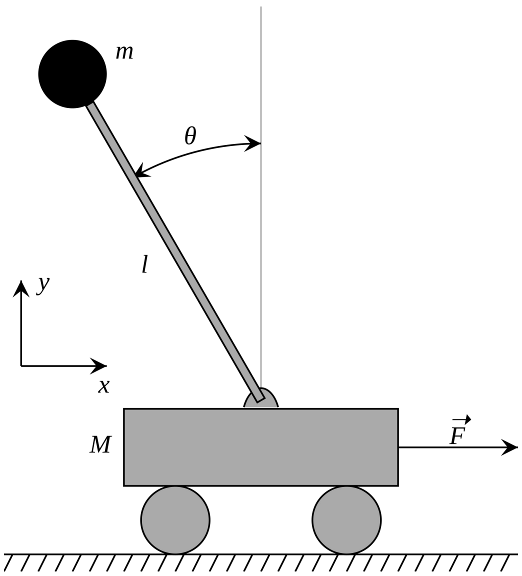
Reinforcement Learning

- What makes reinforcement learning different from other machine learning paradigms?
- There is no supervisor, only a **reward** signal
- **Feedback can be delayed**
  - Time really matters (sequential, non i.i.d data)
- Agent's **actions affect the environment**
  - subsequent data received by the algorithm

Examples

- Chess Play
  - Master players choose the next move based on immediate return and planning, i.e., anticipation of possible replies and counterreplies
- Cleaning robot
  - Decide whether to further explore the space for more trash to collect or start trying to find its way back to the recharging station
- Other daily life examples?

Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart: +1 or -1
Reward: 1 at each time step if the pole is upright

Training: the episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center

Let's see how it works…

https://towardsdatascience.com/cartpole-introduction-to-reinforcement-learning-ed0eb5b58288

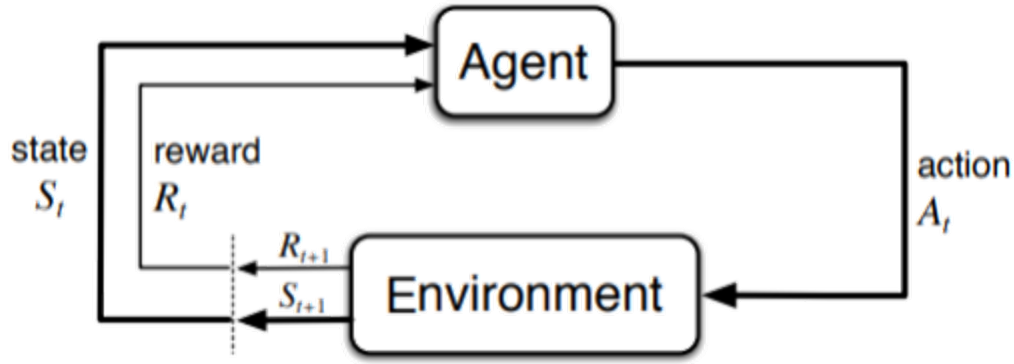
Robot Locomotion

Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Recap

- The state of a system (**environment**) can be modified by some control signals (**actions**) undertaken by controllers (**agents**)
- An action performed in a certain state takes the system in a new state and yields a (positive or negative) reward



- The RL algorithm tries to choose the best action for each system state in order to maximize the "**long-term average reward**"


KEEP
CALM
AND
TAKE A
(SHORT)
BREAK

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

The theoretical foundations: Markov Decision Processes

Processes

- A "process" is a mathematical function that describes the evolution of some entity
- Usually represented as a multidimensional function of one parameter (time): s(t)
- Examples
  - the variation of a room temperature over time
  - the level of water in a lake at August 1st of every year
  - the number of people queueing at ski lift facility
  - the signal attenuation of a wireless link

Stochastic processes

- A stochastic process is a mathematical model that describes a process that can take random values
- More formally, a collection of random variables that is indexed by some mathematical set
- Examples
  - the growth of a bacterial population over time
  - the amplitude of an electric current fluctuating due to thermal noise
  - the number of people infected by a virus every day…

More formally…

- A stochastic processes is then a set S of "random functions" of type $\{S(t,\varpi), t \in T\}$, where T is the *index set* (typically, time or space), ϖ is a sample of a probability space (which embeds the randomicity of the process) and $S(t,\varpi)$ is one specific function (realization) that the process take with given probability $P(\varpi)$
  - Generally, the process is only indicated as $\{S_t\}$

Classification of stochastic processes

- A process is said to be
  - **discrete time** if the index set is numerable: $T=\{t_0,t_1,\ldots\}$
  - **Integer values** if the random functions take values in a numerable set S, i.e., a set whose elements can be associated to (a subset of) the set of integer numbers: $S_t \in S=\{0,\pm 1,\pm 2\ldots\}$
- Examples?

Markov process

- Markovian processes are a family of "memoryless" stochastic processes
- "Memoryless" means that the past "history" of the process up to the current time does not affect the future evolution of the process
- The last observed state of the process is the only one that matters

Markov property in a nutshell

" The future is independent of the past given the present"

More formally

$$\forall t, \forall \{s_{t+1}, s_t, s_{t-1}, \dots, s_0\} \in S$$

it holds

$$\mathbb{P}\Big[\overbrace{\boldsymbol{S_{t+1}} = \boldsymbol{s_{t+1}}}^{\textbf{Future}} \Big| \overbrace{S_t = s_t,}^{\textbf{Present}} \overbrace{\cancel{S_{t-1} = s_{t-1} \dots, S_0 = s_0}}^{\textbf{Past}}\Big] =$$

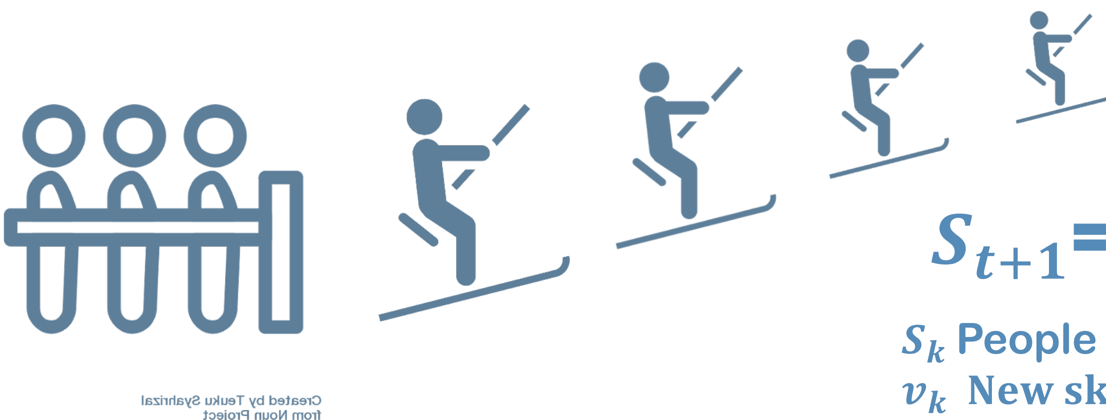
$$\mathbb{P}\Big[\boldsymbol{S_{t+1}} = \boldsymbol{s_{t+1}} \Big| S_t = s_t\Big] = \mathrm{P}_{\boldsymbol{s}_{\mathbf{t}}, \boldsymbol{s}_{\mathbf{t+1}}}$$

Transition probabilities

- Examples?

Example

- A Markov process is fully described by its **current state** and **Transition Probability matrix** $\mathbb{P}$
- Example: queue at ski lift facility



$$S_{t+1} = S_t + v_t - d_t$$

S_k People in queue at time slot k

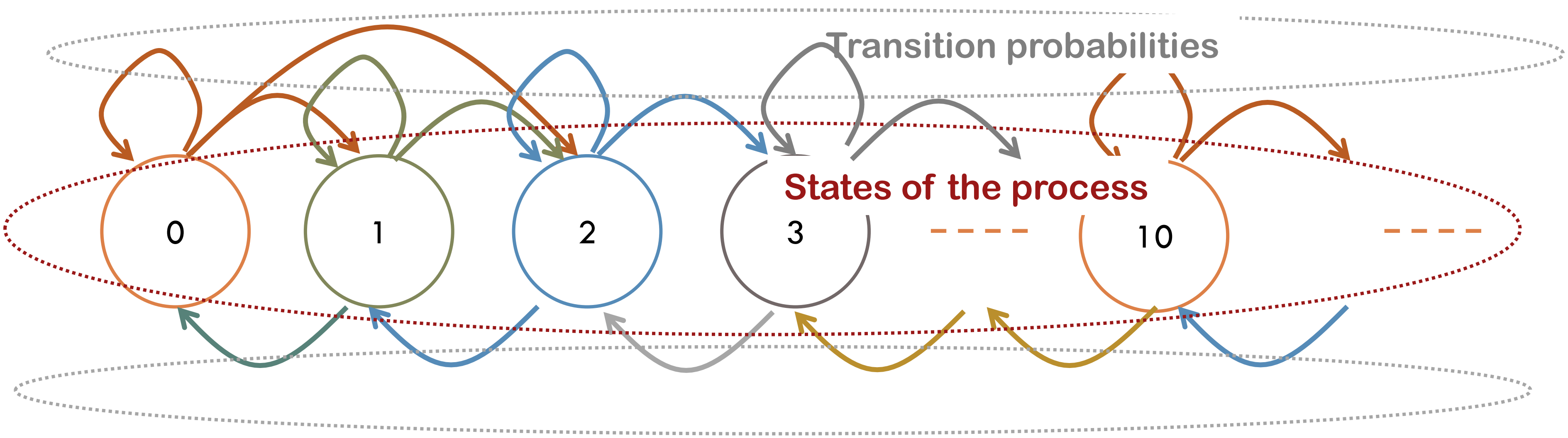
v_k New skiers queuing during slot k

d_k Skiers taking the skilift at slot k $=$ 0 if $S_k = 0$, and 1 otherwise}

Image from: https://thenounproject.com/term/ski-lift/8803/

Transition diagram

$\mathbf{P}[v_k = 0]$ =1/2 $\mathbf{P}[v_k = 1]$ =1/4 , $\mathbf{P}[v_k = 2]$ =1/4

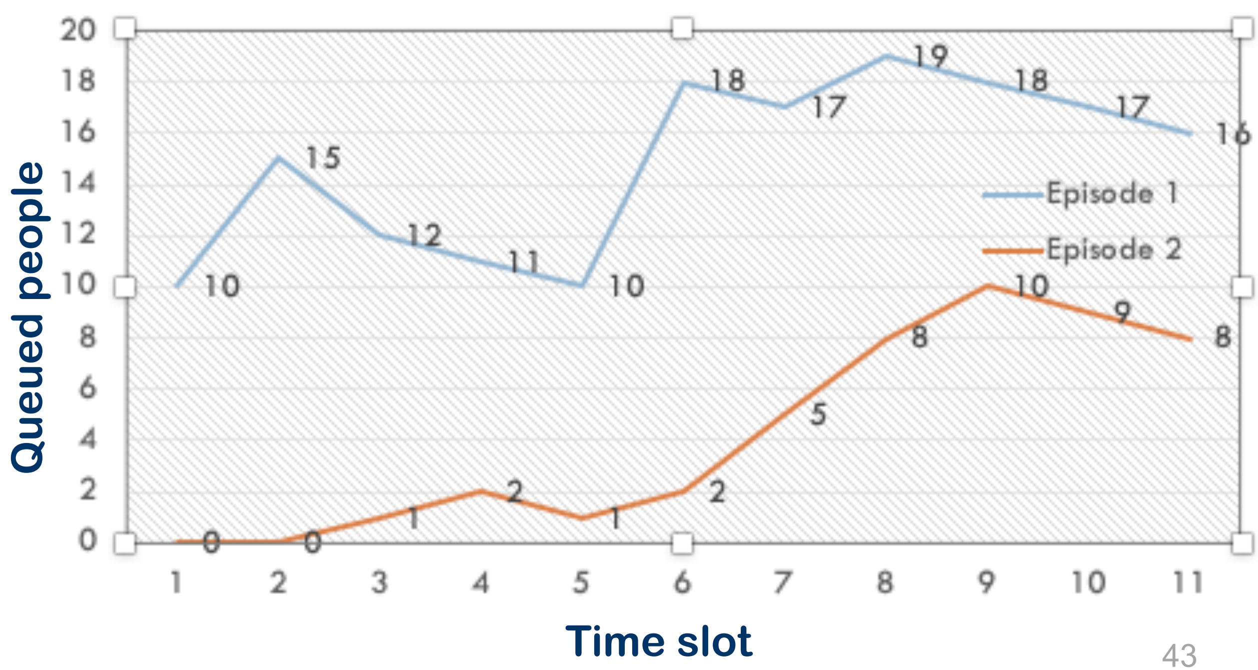


Episodes

- The sequence of actual state values taken by the process in a series of time instants is named a **realization** of the process, or an **episode**
- Example:

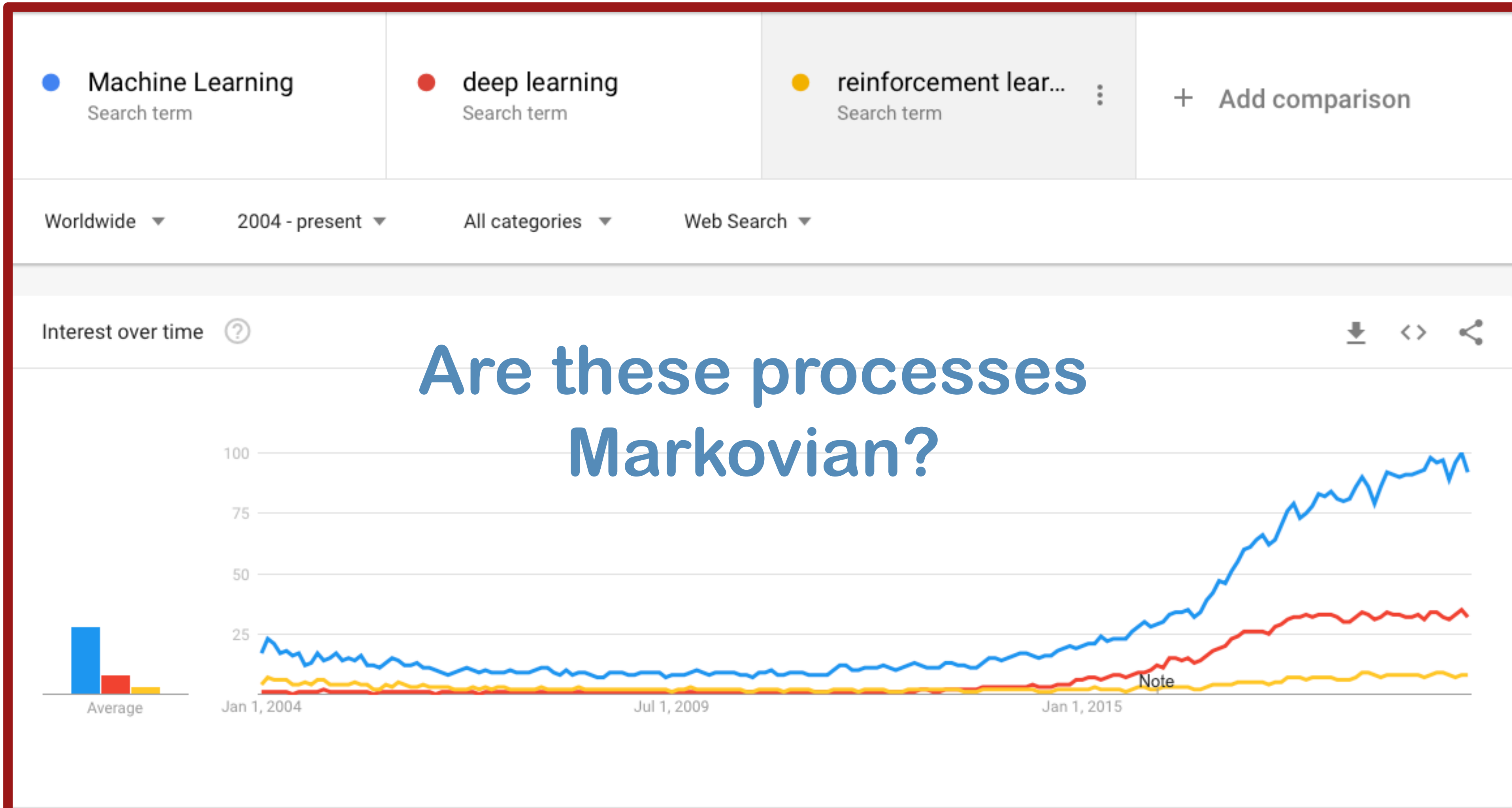
Note:

the two episodes have different probabilities of occurrence



Markov or not Markov?

https://trends.google.it



Are these processes Markovian?

The importance of being earnest…

- It largely depends on:
  - The way we define the “state” of the process
  - The tolerance we accept on our Markovian assumption
- Examples
  - Y_t: temperature in room at time *t* → strong autocorrelation over time interval T → not Markovian
  - The process $S_t = [Y_t, Y_{t-\delta}, Y_{t-2\delta}, \dots Y_{t-m\delta}]$ with mδ>T is “almost” Markovian

Summing up

- State definition is crucial
- State should be “rich” to provide a self-contained description of the system
- State should be “thin” to keep the number of possible values limited
- The choice of a proper state vector is hence **critical** for the proper training of a learning algorithm

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

MARKOV REWARD PROCESS

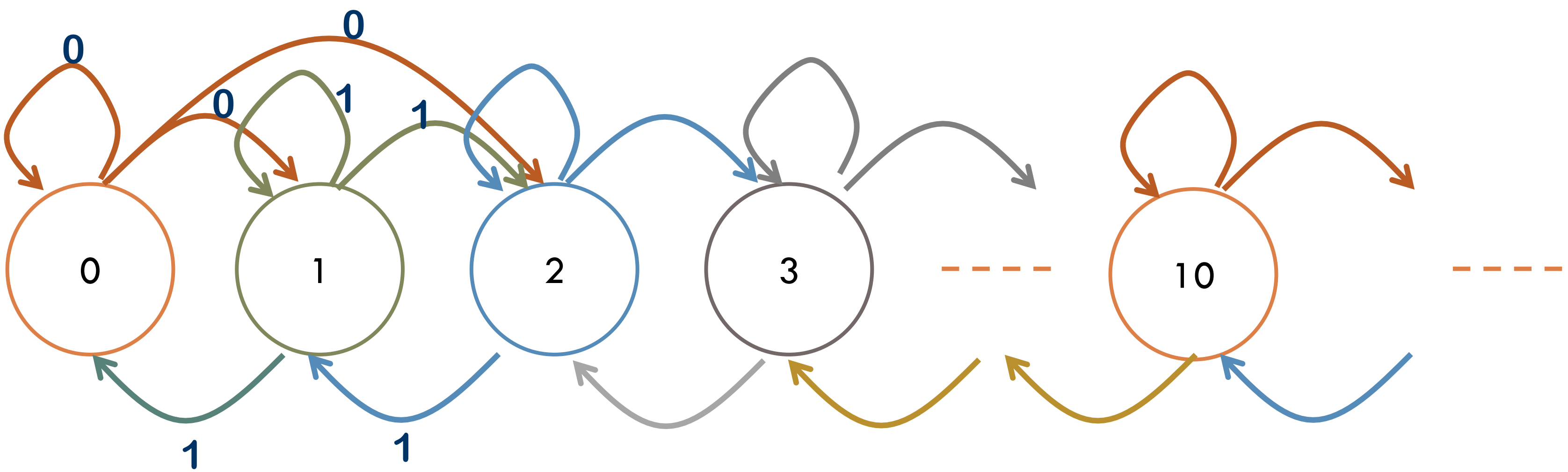
Markov reward process

- A Markov reward process is a Markov process that returns a certain "reward" for each state
- If the state transition depends on a certain action *a*, then $R_t = \mathrm{R}(s, a) \sim \mathrm{P}(\cdot\,|s, a)$where $\mathrm{P}(\cdot\,|s, a)$ is the probability distribution of R_t given the state *s* and the action *a*, *i.e.*,

$$\mathrm{P}(r|s, a) = \mathbb{P}[R_t = r | S_t = s, A_t = a]$$

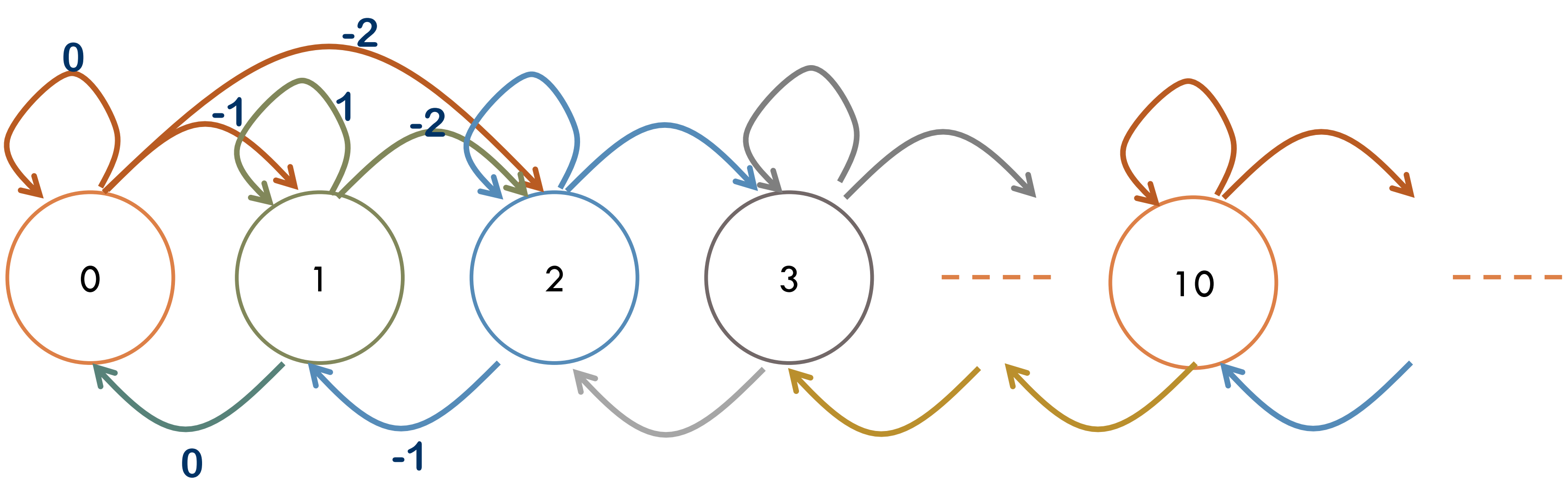
Example of rewards

Reward: 1 if ski lift disk is taken, 0 otherwise



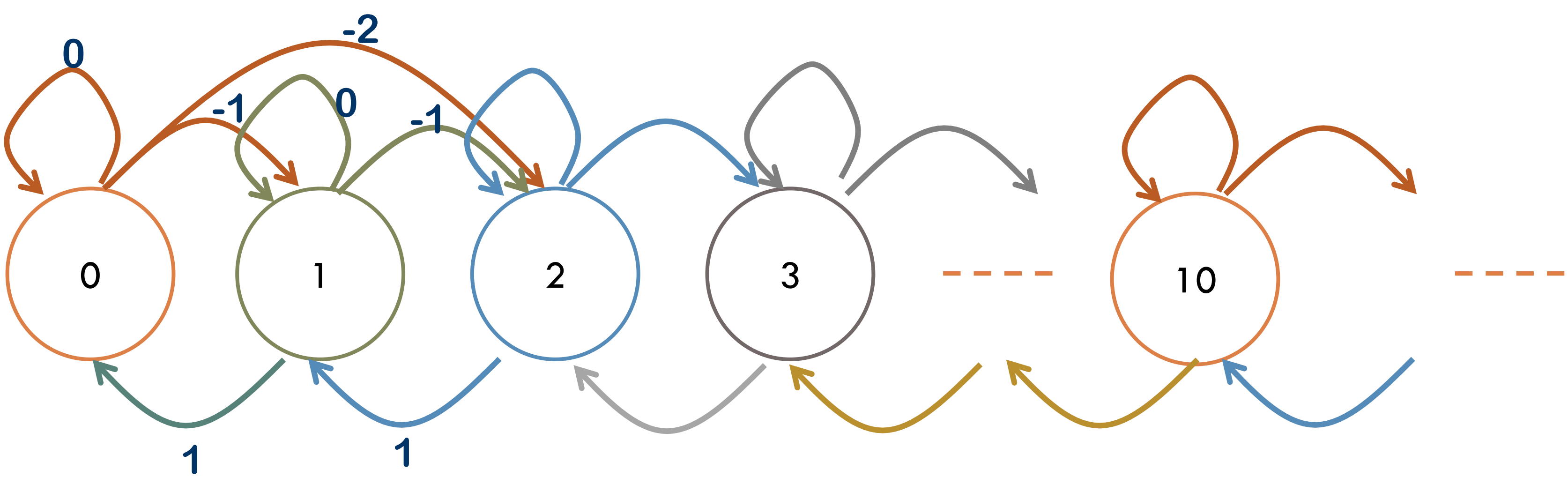
Example of rewards (cont)

Reward: -1 x (number of queued skiers at end of slot)



Example of rewards (cont)

Reward: -1 x (variation of queued skiers)



Markov decision process

- A **Markov Decision Process** (MDP) is a Markov reward process where state transitions and rewards depend on the **actions** taken by a controller

Policy

- For any given state *s* the controller can choose an action A_t in a set $A(s)=\{a_n\}$ of admissible actions, called **Action Space**
- A **policy** at time *t* is the probability distribution of actions for each state
  - $\pi_t(a|s) = \mathbb{P}[A_t = a|S_t = s\ ]$ is the probability that the controller picks action $a \in A(s)$ when the system is in state *s*:
- Each action yields an **immediate reward R_{t+1}** and takes the system to a **new state S_{t+1}**

Policy (cont)

- Given any state and actions *s* and *a*, the probability of each possible pair of next state and reward, *s'* and *r*, is denote

$$\mathrm{P}(s', r | s, a) = \mathbb{P}[S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a]$$

- The expected reward from state-action pair (s,a) is

$$r(s, a) = \mathbb{E}[R_{t+1} | S_t = s, A_t = a] = \sum_r r \sum_{s'} P(s', r | s, a)$$

- The state transition probability is

$$p(s' | s, a) = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] = \sum_r P(s', r | s, a)$$

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

POLICY, VALUE FUNCTION, Q-VALUE

Policy and utility

- The policy affects the evolution of the system state
- Given a policy π, and a starting state *s* we get a sequence of action→ state→ reward
  - $s_0 = s, a_0 \sim \pi(\cdot|s) \rightarrow s_{1,} r_1 \sim P(\cdot,\cdot|s_0, a_0)$
  - $a_1 \sim \pi(\cdot|s_1) \rightarrow s_{2,} r_2 \sim P(\cdot,\cdot|s_1, a_1)$ ...
  - $a_t \sim \pi(\cdot|s_t) \rightarrow s_{t+1}, r_{t+1} \sim P(\cdot,\cdot|s_t, a_t)$, ...
- The accumulation of the rewards over time is a measure of the policy *utility*

How good is a state?

- The **V-function** for a given policy π is the average reward from any state *s* onward:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} R_{t+1} \,\middle|\, s_0 = s, \boldsymbol{\pi}\right]$$

Average over statistical distribution of next states & rewards

Discount factor (γ<1) → future rewards have lower and lower weight

How good is a (state,action) pair?

- The **Q-value (or action-value) function** for a given policy π is a measure of the **utility of a state-action pair**:

$$Q^{\pi}(s,\underbrace{a}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| s_0 = \underbrace{s, a_0 = a}, \boldsymbol{\pi}\right]$$

First action is given

- It is the expected long-term return starting from state *s*, taking action *a*, and thereafter following policy π

Bellman Equation

- The V-function can be expressed in a recursive manner

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \middle| s_0 = s, \boldsymbol{\pi}\right]$$

$$V^{\pi}(s) = \mathbb{E}\left[R_1 + \gamma \sum_{k=0}^{\infty} \gamma^k R_{k+2} \middle| s_0 = s, \boldsymbol{\pi}\right]$$

$$\boxed{V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^{\pi}(s')]}$$

Recap (cont)

- An **MDP** is defined by the tuple: $(S, A, R, \mathbb{P}, \gamma)$
  - S: set of possible states
  - A: set of possible actions
  - R: distribution of reward given (state, action) pair
  - $\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
  - γ: discount factor
- A policy is the probability distribution of actions given state
  - $\pi(a|s) = \Pr[A_t=a|S_t=s]$

Recap

- Each action yields an **immediate reward**
- For a given policy π,
  - The value $V^{\pi}(s)$ of a state *s* is the **aggregate long-term reward** which will be accumulated from that state onwards
  - The **Q-value** $Q^{\pi}(s,a)$ of a *state-action* pair (*s*,*a*) is the **aggregate long-term reward** from state *s*, given that the next action is *a*


KEEP
CALM
AND
TAKE A
(SHORT)
BREAK

Example

Recycling Robot (RR)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Ex RR: problem statement

- Consider a mobile robot that collects cans for recycling
- The robot is battery-power and the battery can be in two states: **high** or **low**
- The robot can perform three different actions:
  - Search for cans
  - Wait for someone to bring it a can
  - Recharge its battery from the dock station

Ex RR: problem statement(cont)

- The probability to collect a can in a certain time interval is
  - r_{search} when searching
  - $r_{wait} < r_{search}$ when waiting
  - 0 when recharging or out of battery
- When moving, the battery level changes
  - from high to high with probability: α
  - from high to low with probability: $1- \alpha$
  - from low to low with probability: β
  - from low to empty with probability: $1-\beta$
- After recharging, the battery goes back to high

Ex RR: MDP model

- State: ?
  - Battery level: S={high, low}
- Action set?
  - A(high) = {search, wait}
  - A(low) = {search, wait, recharge}
- Rewards?
  - R(high,search) = r_{search}
  - R(high,wait) = r_{wait}
  - R(low,search) = $\beta\, r_{search} + (1-\beta)(-3)$
  - R(low,wait) = r_{wait}
  - R(low,recharge) = 0

Ex RR: transition graph

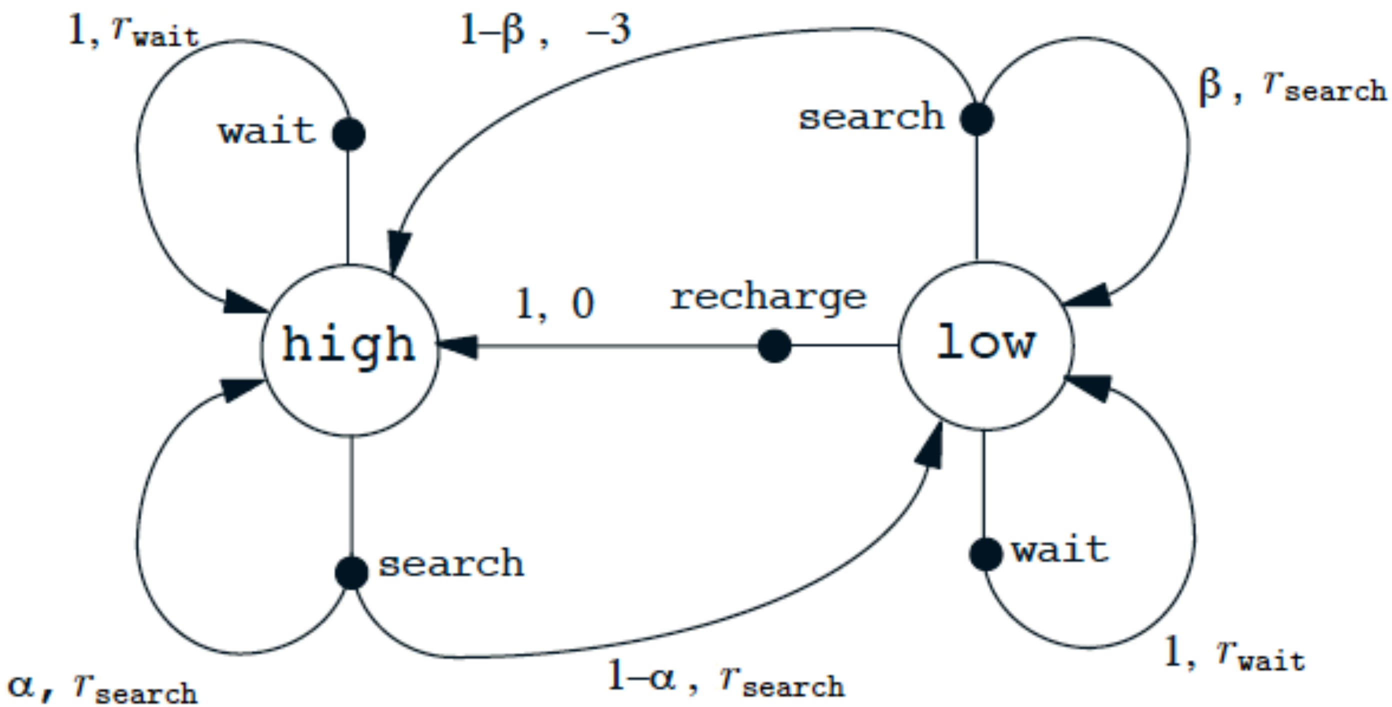


Figure 3.3: Transition graph for the recycling robot example.

"Reinforcement Learning: An Introduction" Second edition, in progress, Richard S. Sutton and Andrew G. Barto 2014, 2015
https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

Iterative Policy Evaluation

- Given a policy, the function V(s) can be obtained by solving the Bellman Ford equations, or through *Iterative Policy Evaluation*:
  - Start from arbitrary (but reasonable) V(s)
  - Apply recursively the Bellman equation to update the V(*s*) value for each *s* or (*s*,*a*) pair:

$$V_{i+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} P(s',r|a,s)\,[r + \gamma\, V_i(s')]$$

  - Repeat until convergence
    - More practically, when $\max_{s \in S} |V_{i+1}(s) - V_i(s)| < \varepsilon$

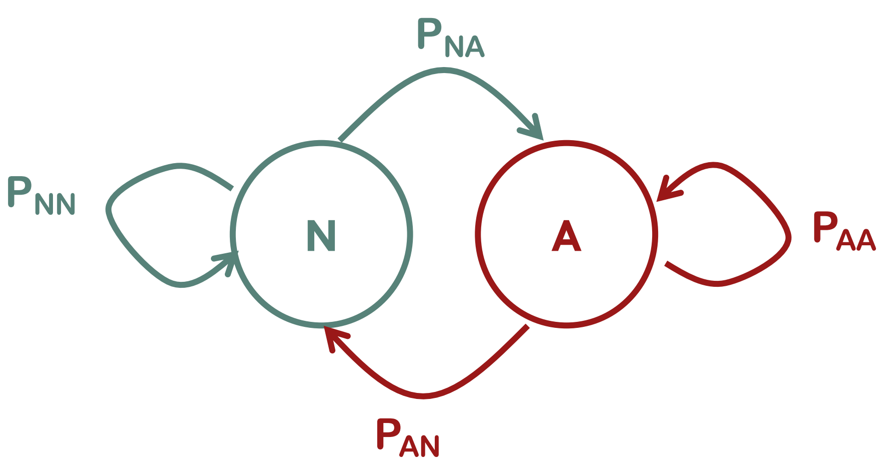
Example

Sensing Strategy (SS)

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Ex SS: problem statement

- A sensor node needs to report its measurements to a control station
- The sensor can work in two conditions: *Normal* (N) or *Alarm* (A)
  - transitions occur as for a **Gilbert model**



Ex SS: problem statement(cont)

- At each slot, the node can either **Transmit** a packet or remain **Idle**
- During "Alarm" periods, packets have **high priority** and should be delivered with max probability
- High priority transmission drains *k=1 or k=2* quanta of energy from the battery of the node, with probability 2/3 and 1/3, resp,
  - If the battery charge is lower than k, the tx fails
- Normal packet transmission takes 1 quantum of energy

Ex SS: problem statement(cont)

- Each Idle slot recharges the battery by one quantum of energy
  - The battery has a maximum capacity of 4 quanta of energy
- If the battery depletes, it cannot be recharged and the node stops working forever

Ex SS: problem statement(cont)

- The sensor node knows its operational conditions and battery level
- Find the transmission policy that maximizes the number of transmitted packets and the probability that high priority

Ex SS: MDP model

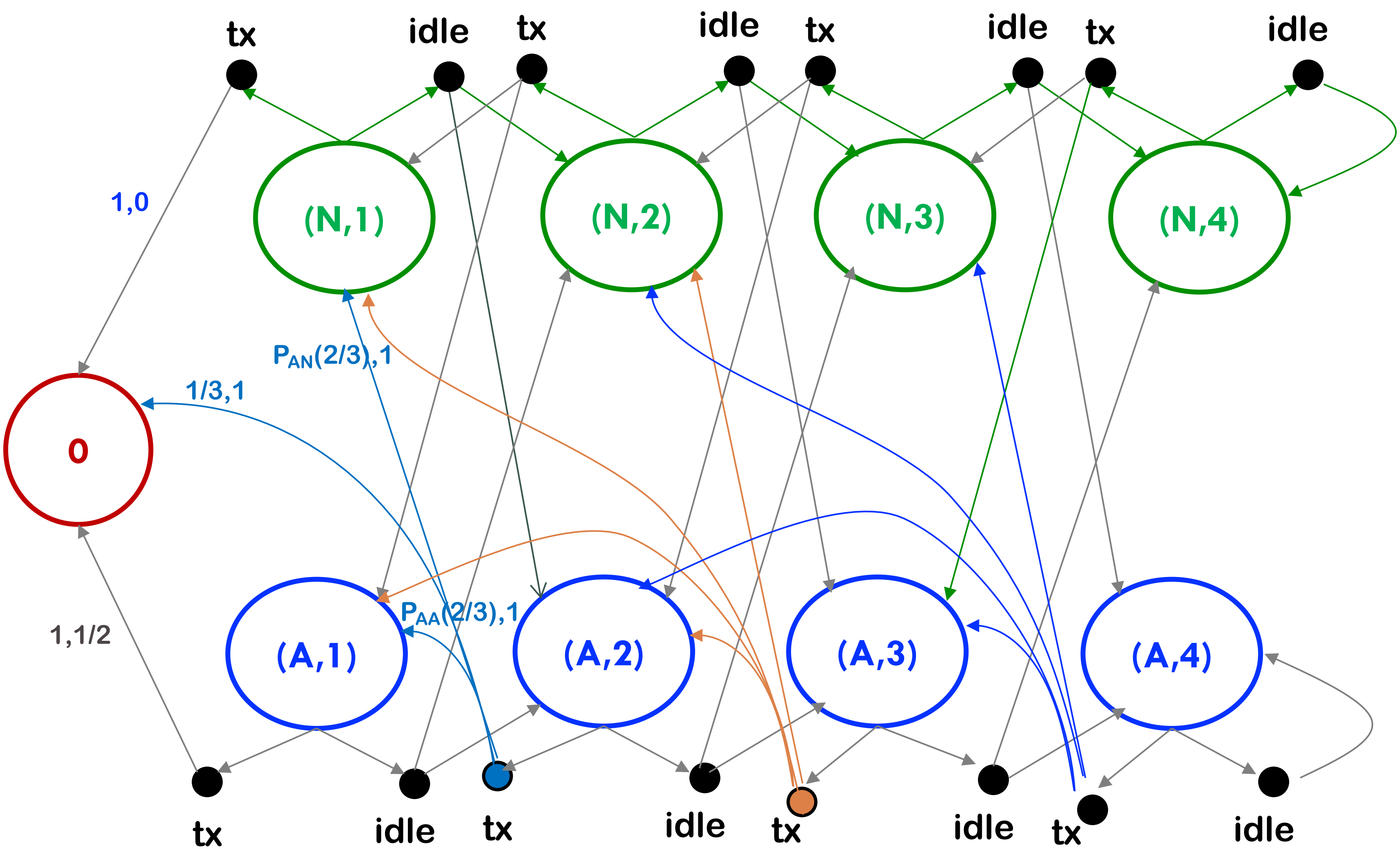
- Take a few minutes to model the problem into an MDP framework
- Which elements do you need to define?
  - System state (Markovian?)
  - Action space
  - State transition probabilities
  - Reward
  - Value function
- What do you need to find?
  - Policy

Ex SS: try by yourself

Ex SS: Solution

- System state:
  - S_t=(operation mode, battery level) or S_t=0 if battery is empty (absorbing state)
- State space:
  - S={0, (N,1), (A,1),…,(N,4),(A,4)}
- Actions: {tx,idle}
- Action space in the different states
  - S_t=(0) → $A(S_t)$={idle}
  - S_t= (*,q) → $A(S_t)$={tx,idle}

Ex SS: transition graph



Ex SS: policy

- Possible policy: $\pi(a|s)$
  - $\pi(idle, 0) = 1$
  - $\pi(tx|(A, 4)) = 1$
  - $\pi(tx|(A, 3)) = 1$
  - $\pi(tx|(A, 2)) = 1$
  - $\pi(tx|(A, 1)) = 0$
  - $\pi(tx|(N, 4)) = 1$
  - $\pi(tx|(N, 3)) = 0.6$
  - $\pi(tx|(N, 2)) = 0$
  - $\pi(tx|(N, 1)) = 0$

Which action is particularly critical?

Ex SS: value function

- $V(0) = 0$
- $V(N,1) = 0 + V(N,2)P_{NN} + V(A,2)\ P_{NA}$
- $V(N,2) = 0 + V(N,3)P_{NN} + V(A,3)\ P_{NA}$
- $V(N,3) = 0.6 + 0.6(V(N,2)P_{NN} + V(A,2)P_{NA}) + 0.4(V(N,4)P_{NN} + V(A,4)P_{NA})$
- $V(N,4) = 1 + V(N,3)P_{NN} + V(A,3)P_{NA}$
- $V(A,1) = 0 + V(N,2)P_{AN} + V(A,2)\ P_{AA}$
- $V(A,2) = 1 + 2/3(V(N,1)P_{AN} + V(A,1)\ P_{AA}) + 1/3V(0)$
- $V(A,3) = 1 + 2/3(V(N,2)P_{AN} + V(A,2)\ P_{AA}) + 1/3(V(N,1)P_{AN} + V(A,1)\ P_{AA})$
- $V(A,4) = 1 + 2/3(V(N,2)P_{AN} + V(A,2)\ P_{AA}) + 1/3(V(N,3)P_{AN} + V(A,3)\ P_{AA})$

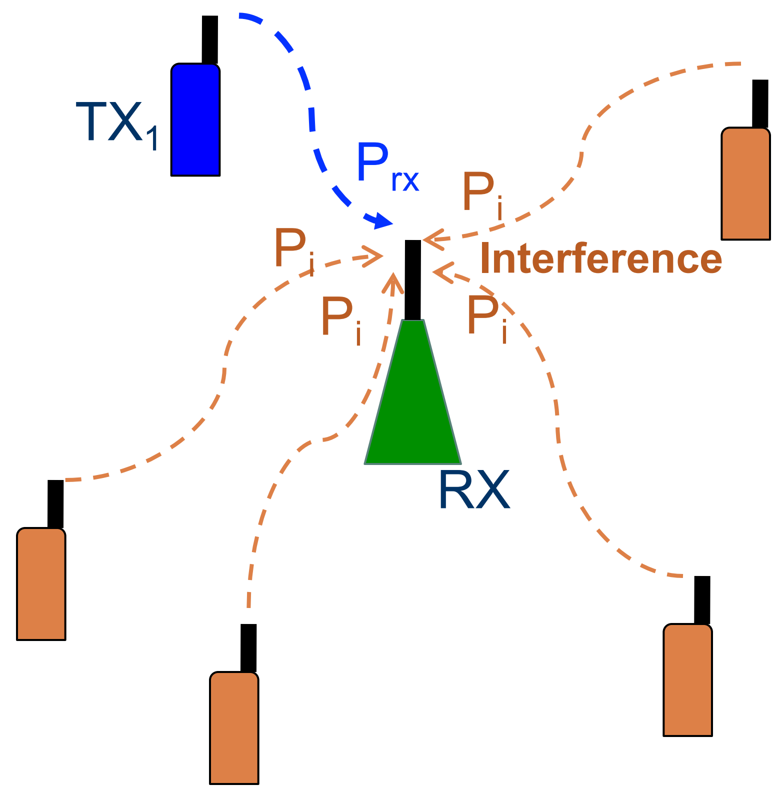
Example

Rate Adaptation (RA) problem

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Ex RA: problem statement

- A wireless node (TX) transmits packets to a receiver (RX)
- Received power P_{rx} depends on channel gain from TX to RX
- A random number *n* of nodes transmit in the background, creating interference power P_i
- The Signal-to-Interference-Ratio (SIR) is given by



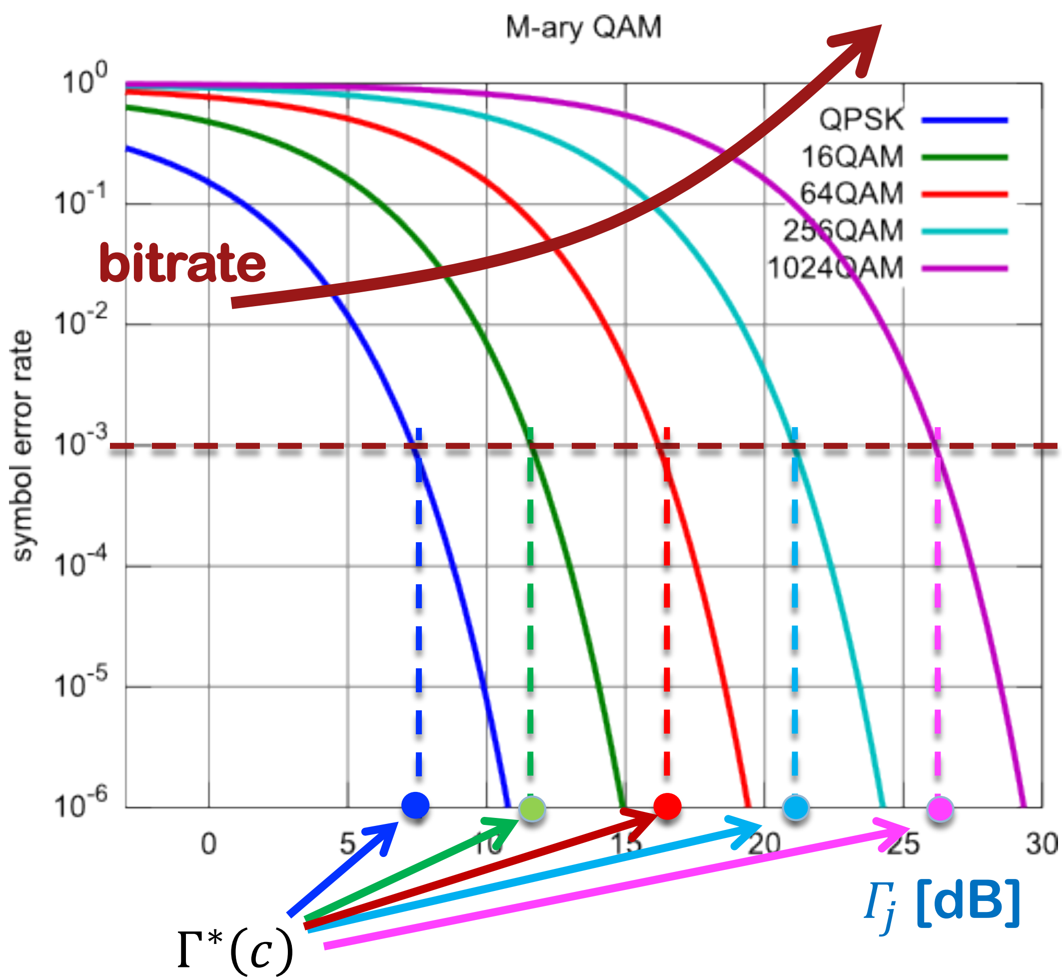
Aggregate interference

$$\Gamma_j(g_j) = \frac{g P_{tx}}{I + \cancel{N_0}} \approx \frac{P_{rx}}{I}$$

Noise power (negligible)

Ex RA: problem statement(cont)

- Transmission rate can be chosen in a set $C=\{c_1,\ldots,c_m\}$
- higher c →
  - higher $\Gamma^*(c)$, i.e., SIR required for correct reception
  - lower transmission time $\tau=L/c$ → lower interference I



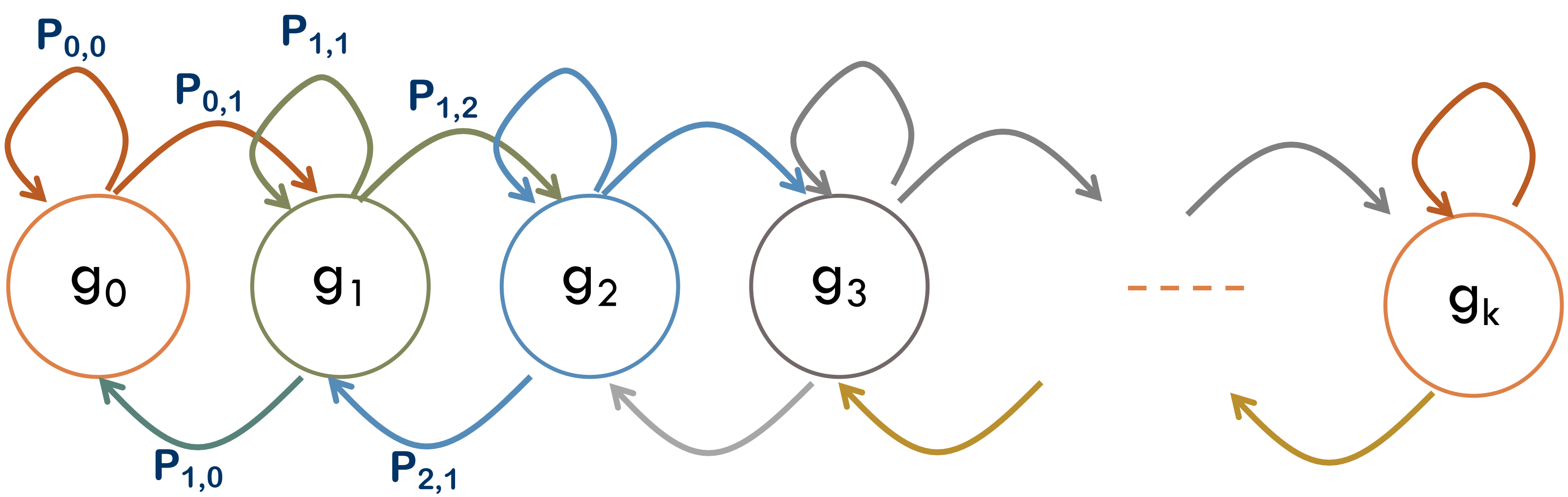
Ex RA: problem statement(cont)

- Problem: find the transmit rate *c* that maximizes the success probability
  - $w(t)$: # of pcks sent up to time t
  - $u(t)$: # of pcks received by RX up to time t

$$\boxed{U = \lim_{t \to \infty} \frac{u(t)}{w(t)}}$$

Ex RA: MDP model

- Assume the channel gain *g* can be modelled as a Markov process with probability transition matrix $\mathbb{P} = [\mathrm{P}_{i,j}]$
- Consider a quantized dB-scale



Ex RA: MDP model (cont)

- The problem can be defined as an MDP
- For each node
  - State at time *t*: channel gain *g*
  - Action space in state *g:* data rates $\{c_1, c_2, \ldots, c_m\}$
  - Reward given by (*g*,*c*): 1 if the SINR is above the reception threshold for *c*, 0 otherwise

Ex RA: MDP model (cont)

- Given a certain policy $\pi(c|g)$ we have
- Reward:

$$R_{t+1} = \chi\left\{\Gamma(g) = \frac{g\mathrm{P}_{tx}}{N_0 + I} > \Gamma^*(c)\right\} = \begin{cases} 1, & if \;\; \Gamma(g) > \Gamma^*(c) \\ 0, & if \;\; \Gamma(g) \leq \Gamma^*(c) \end{cases}$$

- Note:
  - the aggregate interference *I* depends on the number of transmissions that overlap with the target one
  - This number is proportional to the packet transmission time at bitrate *c*: *L/c*
  - R_t is hence random, but given (*g*,*c*) the probability distribution P(-|g,c) is fixed (but maybe unknown)

Ex RA: transition graph?

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

OPTIMAL POLICY

The optimal policy π^*

- The optimal policy π^* maximizes the average value function of all states

$$\boxed{\pi^* = \arg\max_{\pi} \mathbb{E}_s[V^{\pi}(s)]}$$

- An approach to find the optimal policy is to express the Q-values in a recursive manner

Dynamic programming

Bellman Optimality Equation

- Given the optimal policy π* we have

$$V^*(s) = \mathbb{E}_a[Q^*(s,a)] = \sum_{a \in A(s)} \pi^*(a|s) Q^*(s,a)$$

- from which

$$Q^*(s,a) = \mathbb{E}\left[R_t + \gamma \sum_{s'} \mathbb{P}(s'|s,a) V^*(s')\right]$$

$$= \mathbb{E}\left[R_t + \gamma \sum_{s'} \sum_{a'} \pi(a'|s')\, P(s'|s,a) Q^*(s',a')\right]$$

Bellman Optimality Equation

- But since π* is optimal, then π*(a'|s')=1 if and only if a' is the optimal action from s'
- We hence have

$$Q^*(s,a) = \mathbb{E}_{\pi^*}\left[r + \gamma \max_{a'} Q^*(s',a')\,|s,a\right]$$

- which is the **Bellman optimality equation**
- Similarly, we get

$$V^*(s) = \max_{a \in A(s)} \mathbb{E}[R_{t+1} + \gamma V^*(s')|s,a]$$

Optimal policy given Q-values

- If the optimal Q-values $\{Q^*(s,a)\}$ are known for each (s,a) pair, then the optimal policy π^* corresponds to taking for each state *s* the action a_s that maximizes $Q^*(s,a)$:

$$\pi^*: \forall s, \pi^*(a_s|s) = 1 \text{ iff } a_s = \arg\max_a Q^*(s,a)$$

Policy Improvement

- The optimal functions V*(s) and Q*(s,a) can be obtained by solving the Bellman Optimality equations, or through *Iterative Policy Improvement*:
  - Start from arbitrary (but reasonable) policy
  - Apply recursively the Bellman Optimality equation to update the V-function:

$$V_{i+1}(s) = \sum_{s',r} P(s', r \mid a, s)\left[r + \gamma V_{\pi_i}(s')\right]$$

  - Update $\pi_i = \arg\max \sum_{s',r} P(s', r \mid a, s)\left[r + \gamma V_{\pi_i}(s')\right]$
  - Repeat until convergence

Policy Iteration

- **Policy iteration**: concatenate Policy Evaluation and Policy Improvement methods to progressively approach the optimal policy

$$\pi_0 \to V_0 \Rightarrow \pi_1 \to V_1 \Rightarrow \cdots \Rightarrow \pi^* \to V^*$$

- If $\max_{s \in S} |V_{k+1}(s) - V_k(s)| < \varepsilon \to \pi_k \approx \pi^*$

Policy iteration algorithm

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

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\big[r + \gamma V(s')\big]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

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


KEEP
CALM
AND
ENJOY
LUNCH

Example RR (cont)

Let's try with the Recycling Robot problem

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Es RR: optimal policy

- States: h=high, l=low,
- Actions: s=search, w=wait, re=recharge
- Bellman optimality equation

$$
\begin{aligned}
v_*(\mathtt{h}) &= \max\left\{\begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h})+\gamma v_*(\mathtt{h})]+p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l})+\gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h})+\gamma v_*(\mathtt{h})]+p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l})+\gamma v_*(\mathtt{l})] \end{array}\right\} \\
&= \max\left\{\begin{array}{l} \alpha[r_{\mathtt{s}}+\gamma v_*(\mathtt{h})]+(1-\alpha)[r_{\mathtt{s}}+\gamma v_*(\mathtt{l})], \\ 1[r_{\mathtt{w}}+\gamma v_*(\mathtt{h})]+0[r_{\mathtt{w}}+\gamma v_*(\mathtt{l})] \end{array}\right\} \\
&= \max\left\{\begin{array}{l} r_{\mathtt{s}}+\gamma[\alpha v_*(\mathtt{h})+(1-\alpha)v_*(\mathtt{l})], \\ r_{\mathtt{w}}+\gamma v_*(\mathtt{h}) \end{array}\right\}.
\end{aligned}
$$

$$
v_*(\mathtt{l}) = \max\left\{\begin{array}{l} \beta r_{\mathtt{s}}-3(1-\beta)+\gamma[(1-\beta)v_*(\mathtt{h})+\beta v_*(\mathtt{l})] \\ r_{\mathtt{w}}+\gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array}\right\}.
$$

Example RA (cont)

Let's try with the Rate Adaptation (RA) problem

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Ex RA (cont)

- Immediate reward:

Random because of interference

$$r \sim \mathrm{R}(g,c) = \begin{cases} 1, & \Gamma(g) > \Gamma^*(c) \\ 0, & \Gamma(g) \le \Gamma^*(c) \end{cases}$$

- Q-value:

$$Q^\pi(g,c) = \mathbb{E}\left[\mathrm{R}(g,c) + \gamma \sum_{g'=\mathrm{g}_0}^{\mathrm{g}_\mathrm{k}} \mathbb{P}(g'|g,c) V^\pi(g')\right]$$

$$= \Pr[\Gamma(g) > \Gamma^*(c)] + \gamma\, \mathbb{E}\left[\sum_{g'=\mathrm{g}_0}^{\mathrm{g}_\mathrm{k}} \mathbb{P}(g'|g,c) V^\pi(g')\right]$$

- V-function:

$$V^\pi(g) = \mathbb{E}_c[Q^\pi(g,c)] = \sum_{c_h} \pi(c_h|g) Q^\pi(g,c_h)$$

Ex RA (cont)

- **Note**: the next state (channel gain) does not depend on the chosen action (bitrate) →
  - $\mathbb{P}(g'|g,c) = \mathbb{P}(g'|g) = P_{g,g'}$
- In this case, the future rewards do not depend on the current action
- The Bellman equation yields
  - $Q^*(g,c) = \bar{R}(g,c) + \gamma \mathbb{E}_{g' \sim \mathbb{P}(\cdot|g)}\left[\max_{c'} Q^*(g',c')\right]$

Ex RA (cont)

- Since the right-most term does not depend on the current action *c*, we have that
  - $\max_{c} Q^*(g,c) = \max_{c} \bar{R}(g,c)$
- with

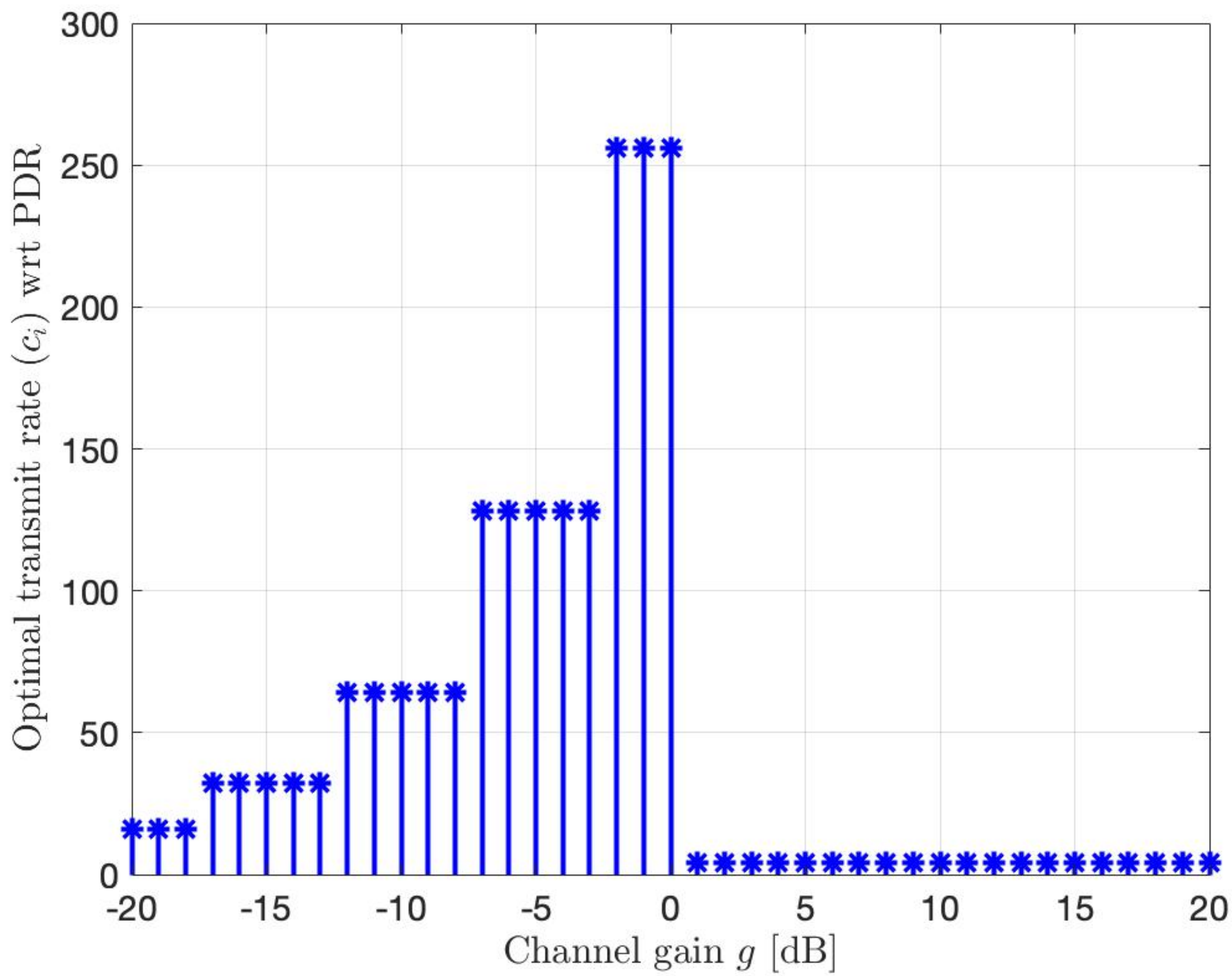
$$\bar{R}(g,c) = \Pr[\Gamma(g) > \Gamma^*(c)] = \Pr\left[n < \frac{g\mathrm{P}_{tx}}{\mathrm{P}_I \Gamma^*(c)}\right]$$

- where *n* is the number of transmissions that interfere with the target one

Ex RA (cont)

- Assuming *n* is Poisson with parameter λL/c we get
- $\bar{R}(g,c) = \sum_{k=0}^{\left\lfloor \frac{g\mathrm{P}_{tx}}{\mathrm{P}_I \Gamma^*(c)} \right\rfloor} \frac{\left(\frac{\lambda L}{c}\right)^k}{k!} e^{-\frac{\lambda L}{c}}$
- Plotting $\bar{R}(g,c)$ vs *c* for different *g* we find the optimal action of each state

Ex RA (cont)



300
250
200
150
100
50
0
-20
-15
-10
-5
0
5
10
15
20
Optimal transmit rate (c_i) wrt PDR
Channel gain g [dB]

Ex RA b

- What if we want to maximize the average throughput?
  - $w(t)$: # of pcks transmitted by TX up to time t
  - $u(t)$: # of pcks received by RX up to time t

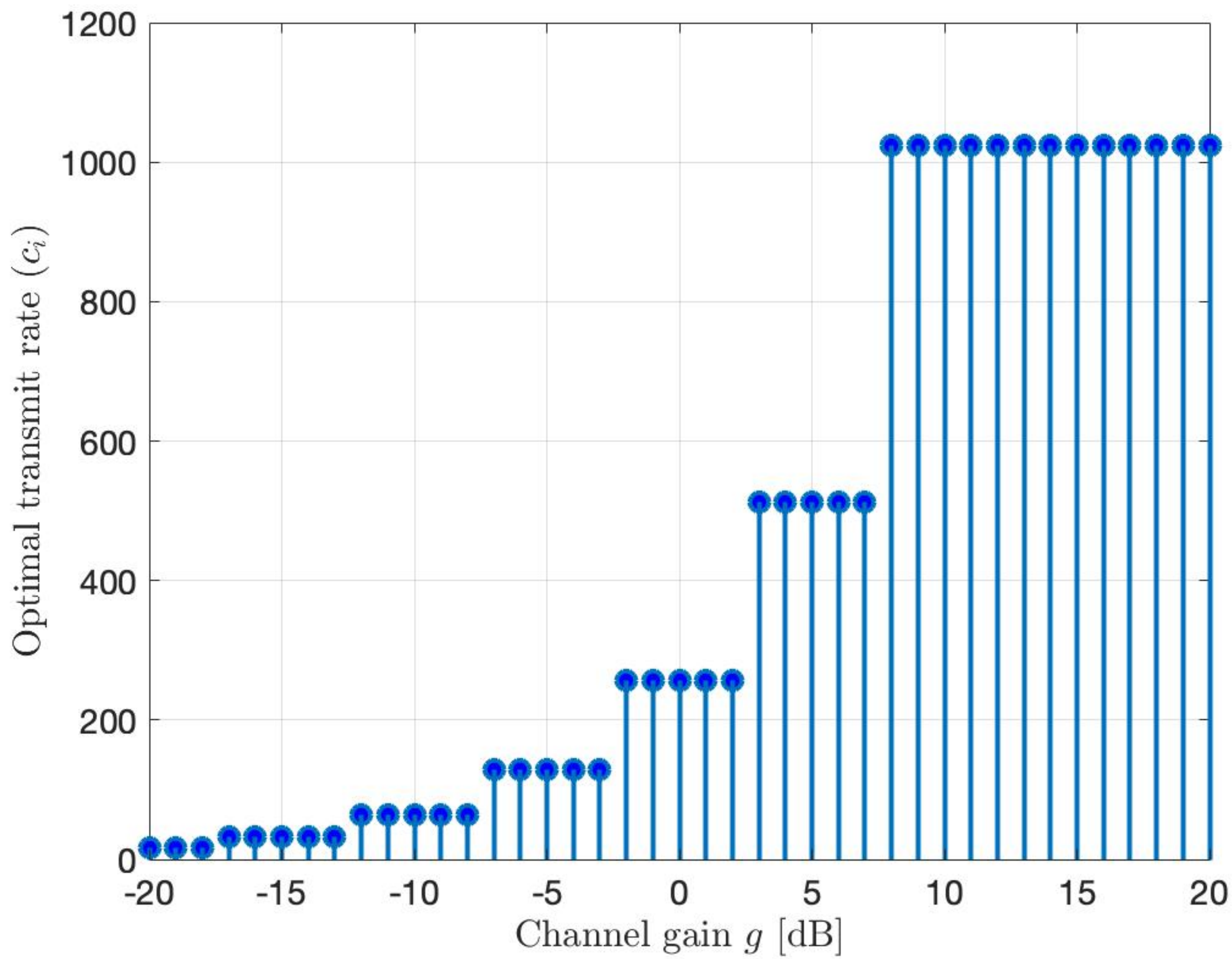
$$\boxed{U = \lim_{t \to \infty} \frac{u(t)}{\sum_{i=1}^{w(t)} 1/c(t)}}$$

Ex RA b (cont)

- Immediate reward:

$$\mathrm{r} \sim \mathrm{R}(g, c) = \begin{cases} c, & \Gamma(g) > \Gamma^*(c) \\ 0, & \Gamma(g) \leq \Gamma^*(c) \end{cases}$$

Ex RA b (cont)



1200
1000
800
600
400
200
0
-20
-15
-10
-5
0
5
10
15
20
Optimal transmit rate (c_i)
Channel gain g [dB]

Piece of cake... Or not?

- Do you see any problem?
- Must compute Q(s,a) for every state-action pair
  - If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Not scalable!**

Curse of dimensionality!

- Furthermore, transition probabilities **p(s',r|s,a) must be known beforehand**

Solution: Reinforcement Learning

- If P(s',r|s,a) is known → Markov Decision Process (MDP)
- If it is not → Reinforcement learning (RL)
- Reinforcement learning:
  - Model-based: Learn a model of P(s',r|s,a) and then solve as MDP
  - Model-free: Learn directly the policy

Generalized Policy Iteration

- Solution methods for both MDP and model-free RL
- Basic idea:
  - **Policy Evaluation:** emulate the system evolution for a few steps always using *policy behavior*
  - **Control:** update control policy at each step based on Q values
  - Periodically, set behavior policy to control policy

Generalized Policy Iteration

- Generalized Policy Iteration
  - **Policy Evaluation**
  - Control

On-Policy & off-policy

- **On-policy** method
  - behavior and control policies are always the same
- **Off-policy** method
  - behavior policy is used for a certain number of steps and only periodically replaced with current control policy
  - Off-policy algorithms have an advantage: they can take more risks during exploration, since mistakes will not propagate to control policy

Policy evaluation

- Action-value function: backup diagram

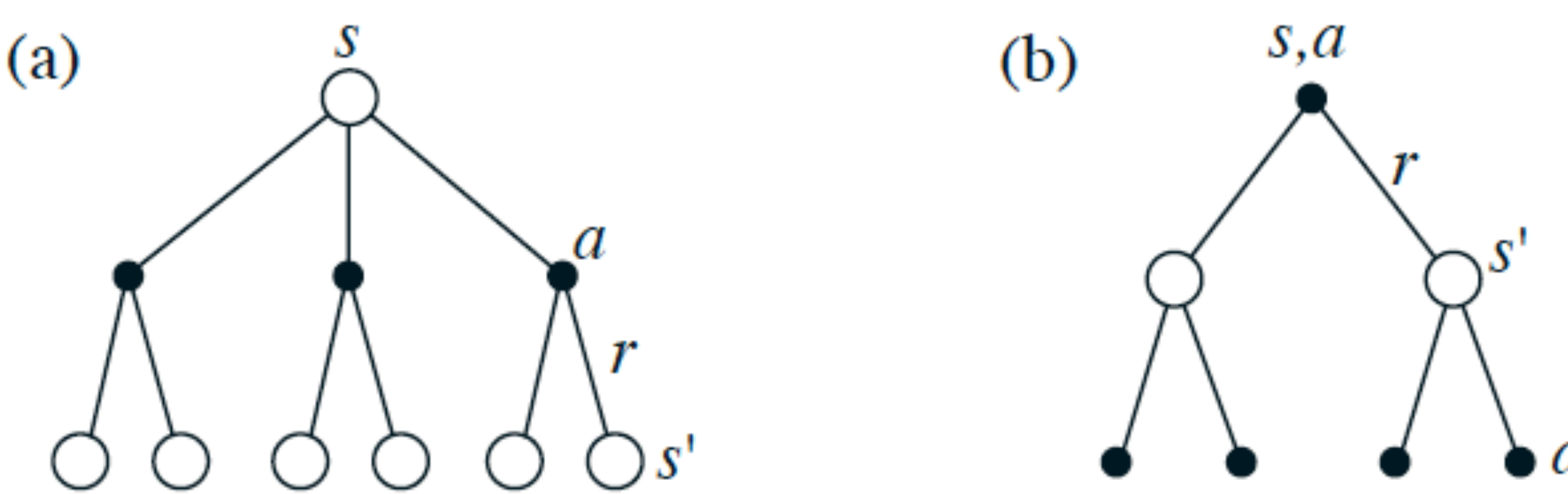
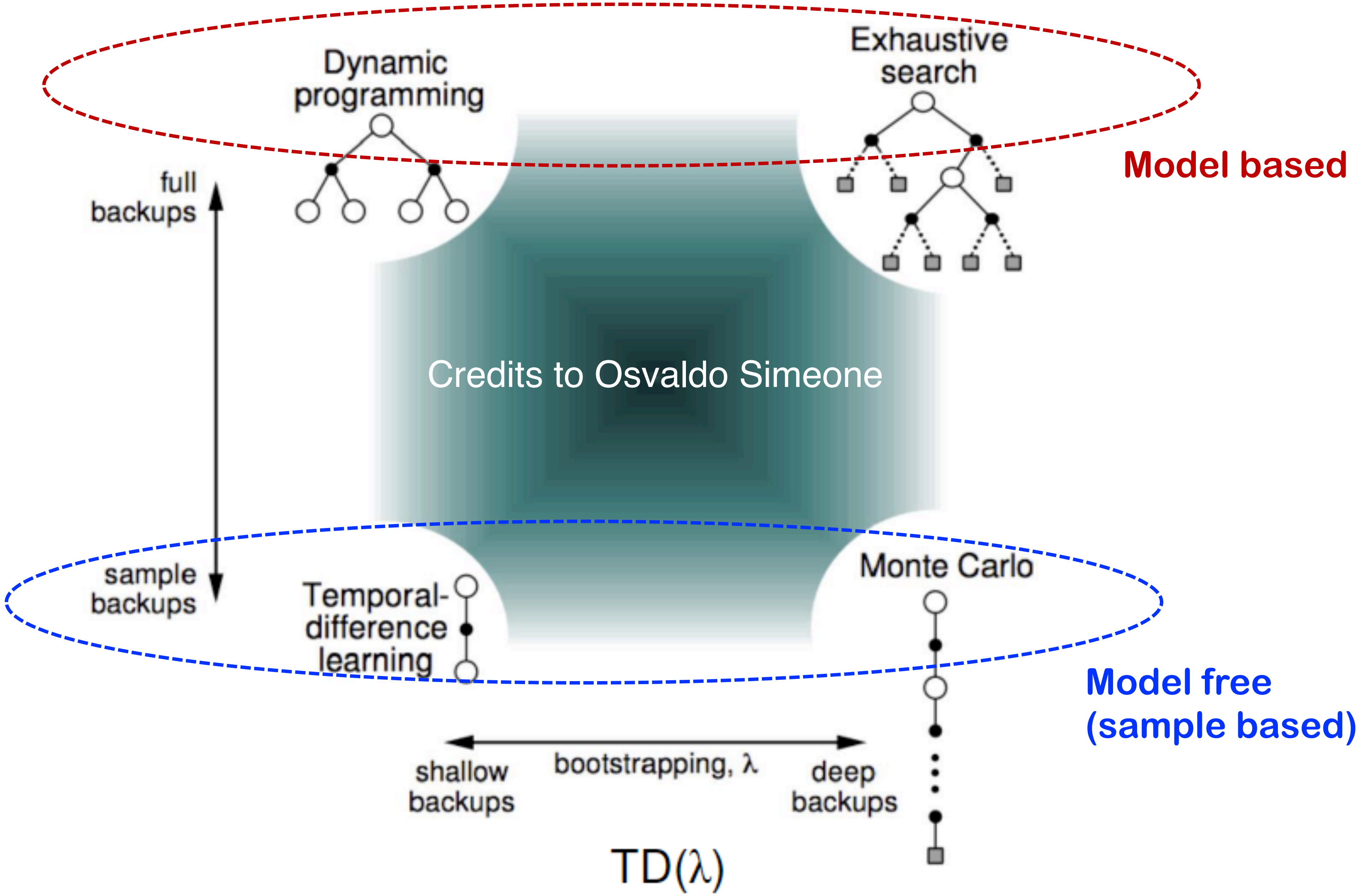


Figure 3.4: Backup diagrams for (a) v_π and (b) q_π.

"Reinforcement Learning: An Introduction" Second edition, in progress, Richard S. Sutton and Andrew G. Barto 2014, 2015
https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

Policy evaluation



Policy evaluation

- Exploration in breadth generally requires the availability of a model
  - For each state → compute average value by considering probability of next state and reward
- Model-free methods estimate policy and probabilities
- Based on
  - Temporal-Difference (online) → $Q(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, a_{t+1})$
- and/or
  - Monte Carlo (offline) → $Q(s_t, a_t) \approx r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$

Generalized Policy Iteration

- Generalized Policy Iteration
  - Policy Evaluation
  - **Control**

Exploration vs exploitation

- **Exploitation**: make the most out of what you know
  - Take actions that maximize return based on current knowledge of Q-values and Value function
- **Exploration**: check other strategies to see whether you can do any better
  - Take actions that are not immediately optimal, but can improve estimate of the long-term returns

Control policies

- There are several well-known control policies
- The most common are **ε-greedy** and **softmax**
- In both cases, there is some randomness to explore the state and action spaces

Epsilon-Greedy Policy

most likely selects the greedy action

Softmax policy

Selects the actions based on their relative Q-values

$$\pi_i(a|s) = \frac{e^{Q_i(s,a)}}{\sum_{a'} e^{Q_i(s,a')}}$$

Curse of dimensionality

- Do you see any problem?
- Must compute Q(s,a) for every state-action pair
  - If state is e.g. current game state pixels, computationally infeasible to compute for entire state space!
- **Not scalable!**

Curse of dimensionality!

- Furthermore, transition probabilities **p(s',r|s,a) must be known beforehand**

Q-Learning

- Q-learning: use a neural network to approximate the action-value function

$$Q^*(s,a) \approx Q(s,a,\Theta)$$

- If the function approximator is a deep neural network → **deep q-learning**!

Q-Learning

- The neural network should find an approximation of the Q-function that satisfies the Bellman equation

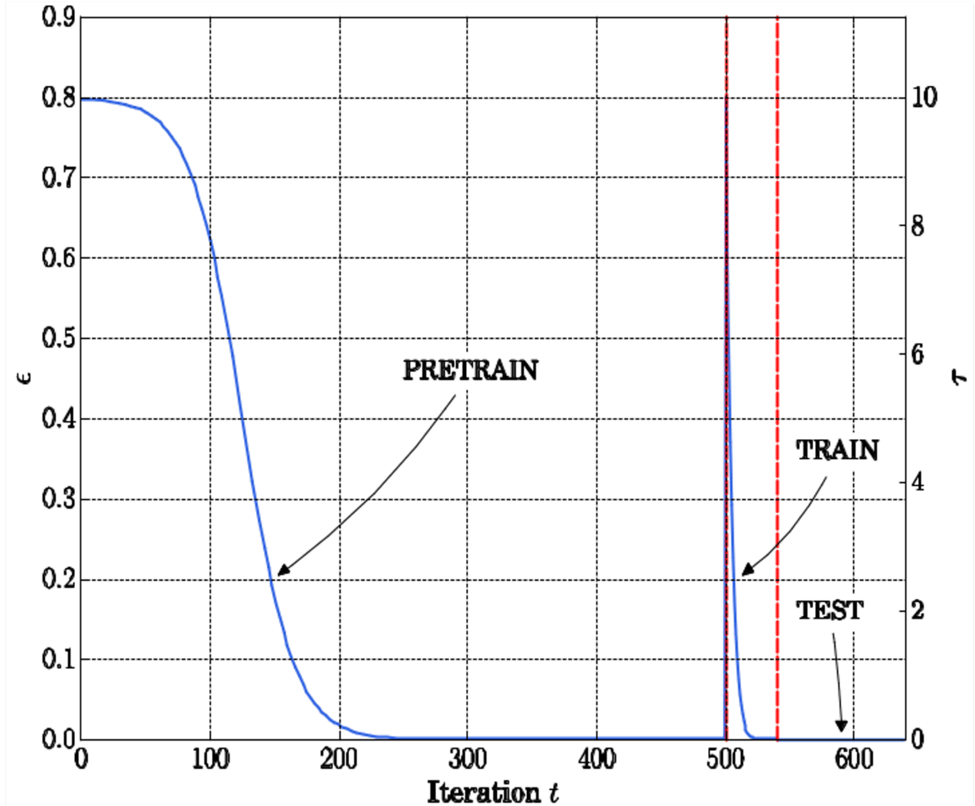
$$Q^*(s,a) = \mathbb{E}_{\pi^*}\left[r + \gamma \max_{a'} Q^*(s',a') \,|s,a\right]$$

- Forwards pass
  - Loss Function: $L_i(\theta_i) = \mathbb{E}_{s,a}[(y_i - Q(s,a;\theta_i))^2]$
  - With $y_i = \mathbb{E}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) \,|s,a\right]$
- Backward pass
  - Gradient update (wrt to Q-function parameters θ):

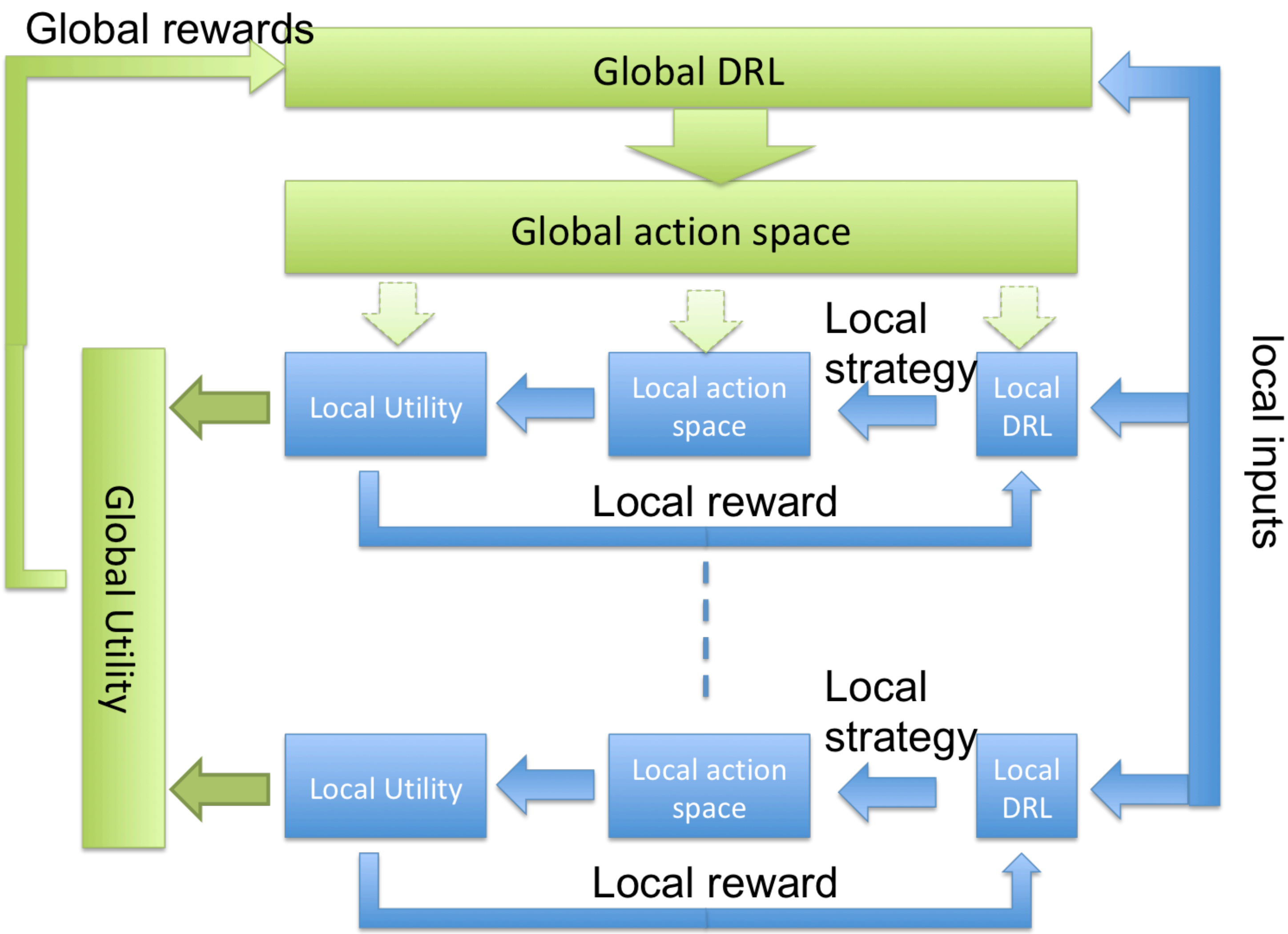
$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a,s'}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right] \nabla_{\theta_i} Q(s,a;\theta_i)$$

The exploration profile

- Exploration is crucial in the initial phases: the agent needs to find out as much as it can from the environment
- If the agent is too greedy in the first episodes, it can get stuck in a local maximum
- A pre-training phase using a simplified environment can help if the real one is unavailable or computationally heavy (or if good performance is needed from the start)



Hierarchical Deep Reinforcement Learning



Main reference

- “Reinforcement Learning: An Introduction” Second edition, in progress, Richard S. Sutton and Andrew G. Barto 2014, 2015, https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf