Fundamentals of **Reinforcement Learning**

Human-Interactive Robot Learning (HIRL)
Silvia Tulli - Kim Baraka - Mohamed Chetouani

Agenda

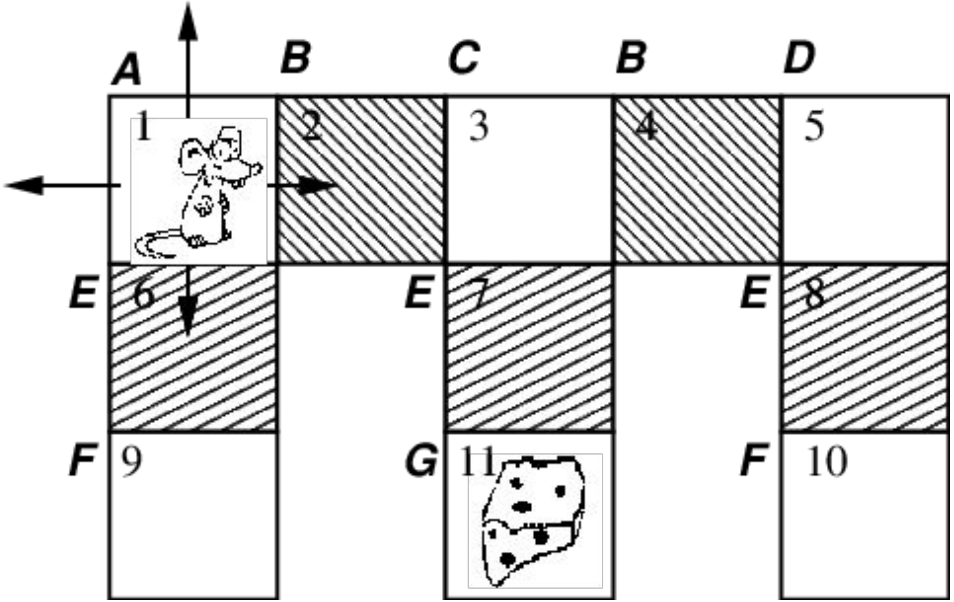
- Recap Questionnaire on Human-Interactive Robot Learning
- Fundamentals of RL Course
- Practice with Jupyter Notebooks
- Final Questionnaire on today's material, both course and practice

Learning Goals

By the end of this lecture, you should be able to:

- Frame a sequential decision making problem as an MDP
- Explain the concept of value function (V) and action function (Q)
- Apply value iteration and policy iteration to solve an MDP
- Explain general RL concepts such as Model-based vs. Model-free RL and the exploration-exploitation tradeoff
- Master the following algorithms
  - Value Iteration
  - Policy Iteration
  - Dynamic Programming
  - Q-learning
  - SARSA

Toy Example of SDM



Sequential Decision Making - Simple Gridworld

- Robot/Mouse (agent)
- Grab the cheese (goal)
- Maze (environment)
- Cells of the maze (possible states)
- Arrows (agent's possible actions)

Noisy environment - if you move forward you do not necessarily go forward

After the learning happens and after we realise how to solve it optimally that would be a potential trajectory when everything is done

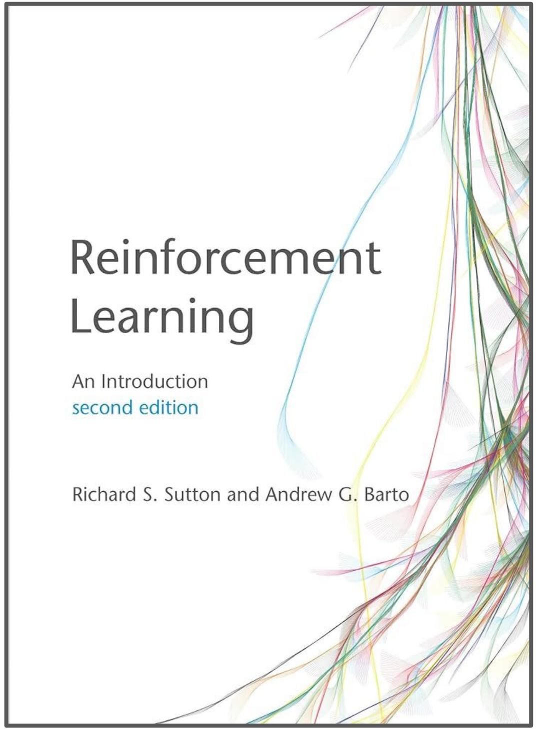
Markov Decision Process

MDP is a model for sequential decision making

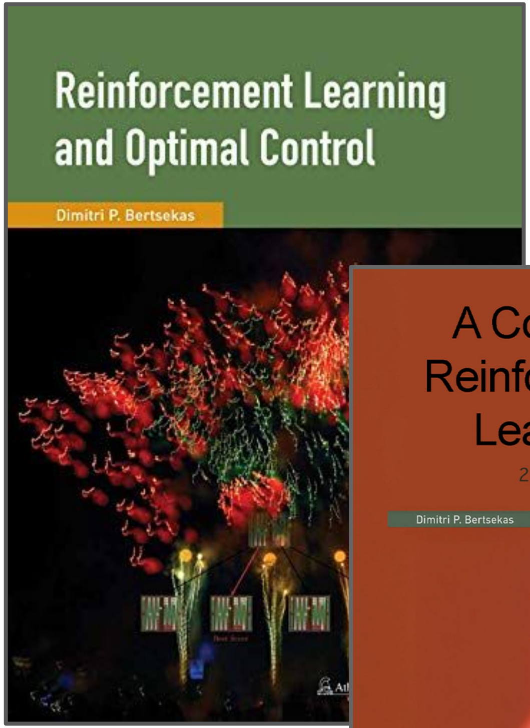
The "Markov" in "Markov decision process" refers to the underlying structure of state transitions that still follow the **Markov property.**

The Markov property means that evolution of the Markov process in the future depends only on the present state and does not depend on past history.

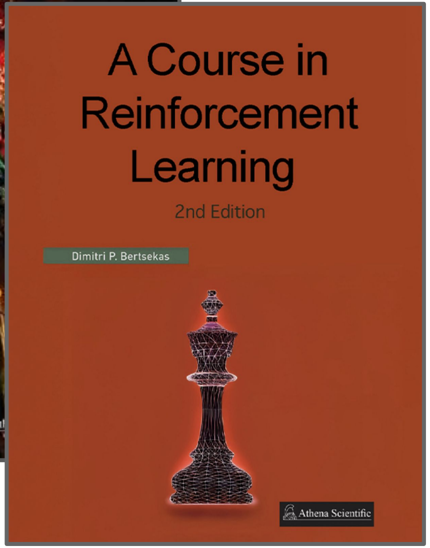
References



Reinforcement Learning: An Introduction, [Richard S. Sutton and Andrew G. Barto (2018)]

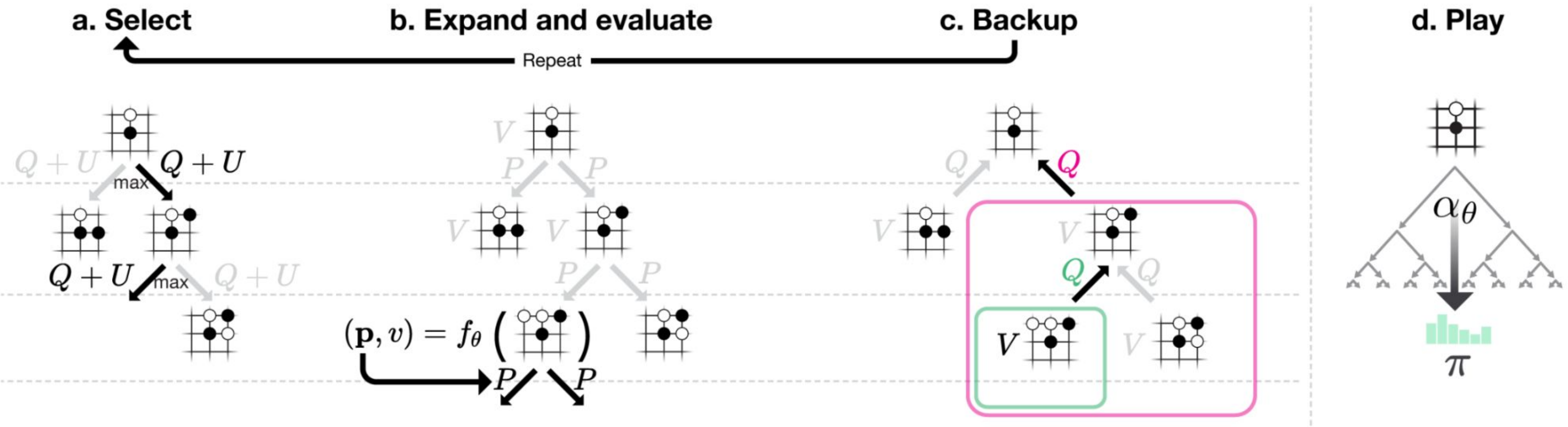


Reinforcement Learning and Optimal Control, Athena Scientific (2019)



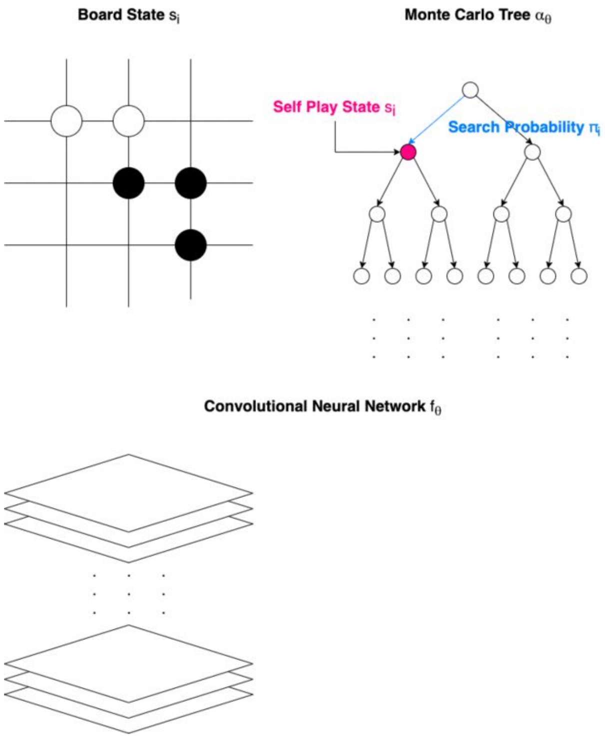
A Course in Reinforcement Learning: 2nd Edition, ([free pdf] Athena Scientific, 2025)

Mastering the game of Go without human knowledge



Monte-Carlo tree search in AlphaGo Zero, Silver et al. 2017, nature.com/articles/nature24270

Mastering the game of Go without human knowledge



Source: medium.com

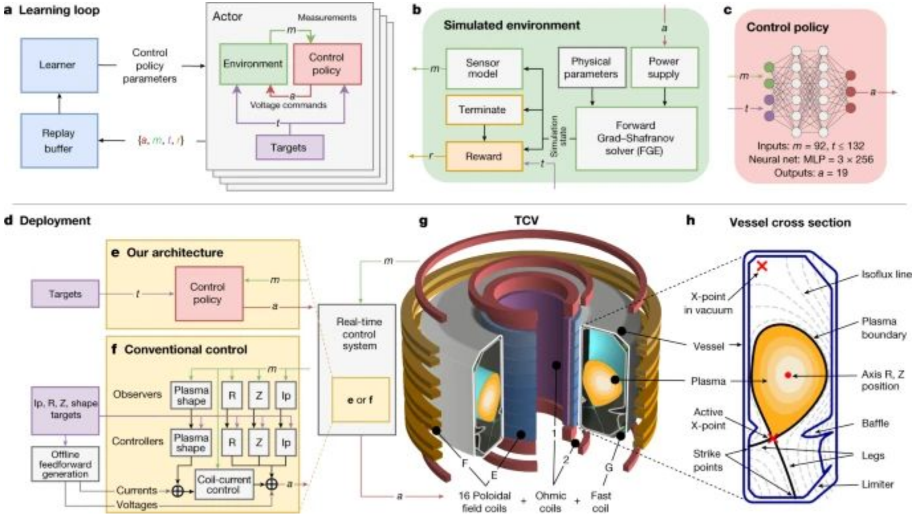
[…] *an algorithm based solely on reinforcement learning,* ***without human data****, guidance or domain knowledge beyond game rules.*

AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration.

Starting ***tabula rasa****, our new program* ***AlphaGo Zero*** *achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.*

Monte-Carlo tree search in AlphaGo Zero, Silver et al. 2017, nature.com/articles/nature24270

Magnetic control of tokamak plasmas through deep RL

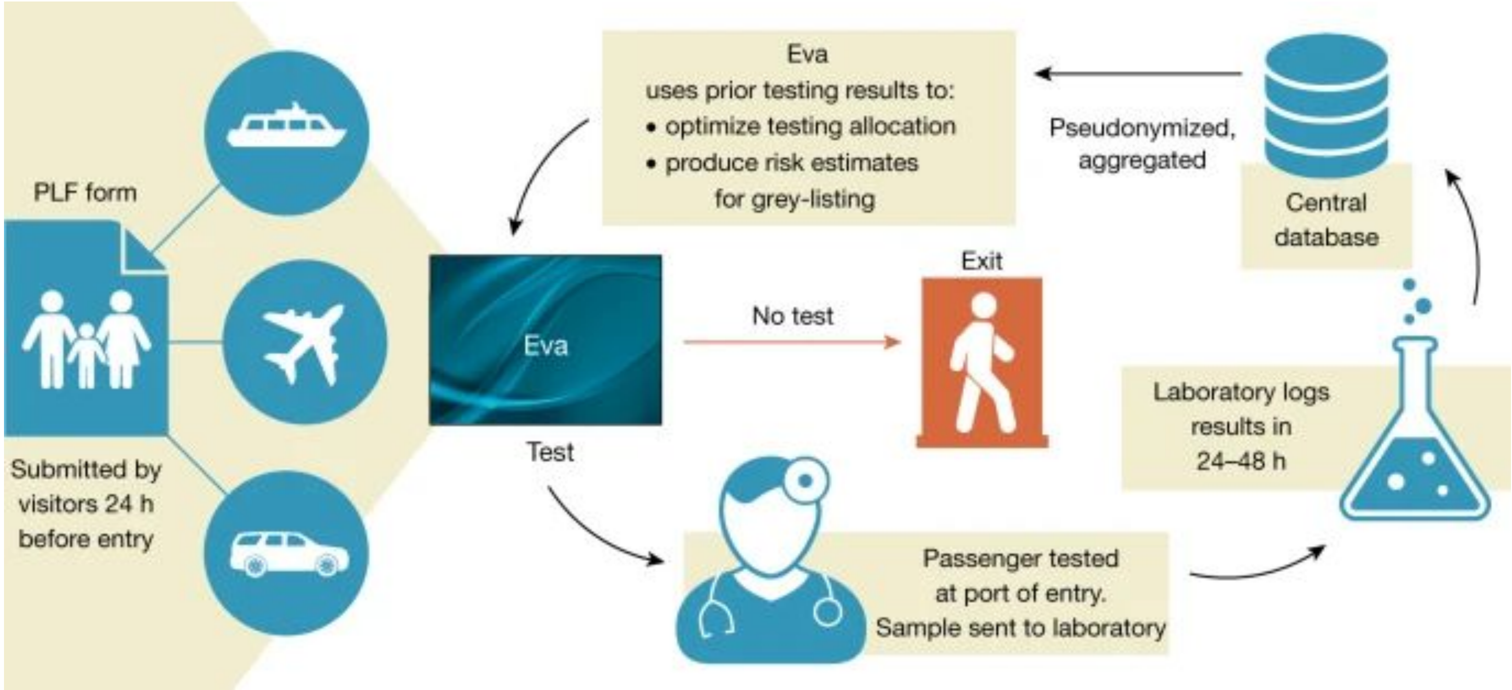


Representation of the components of our controller design architecture

[...] *introduce a previously undescribed architecture for tokamak magnetic controller design that autonomously learns to command the full set of control coils. This architecture meets control objectives specified at a high level, at the same time satisfying physical and operational constraints*

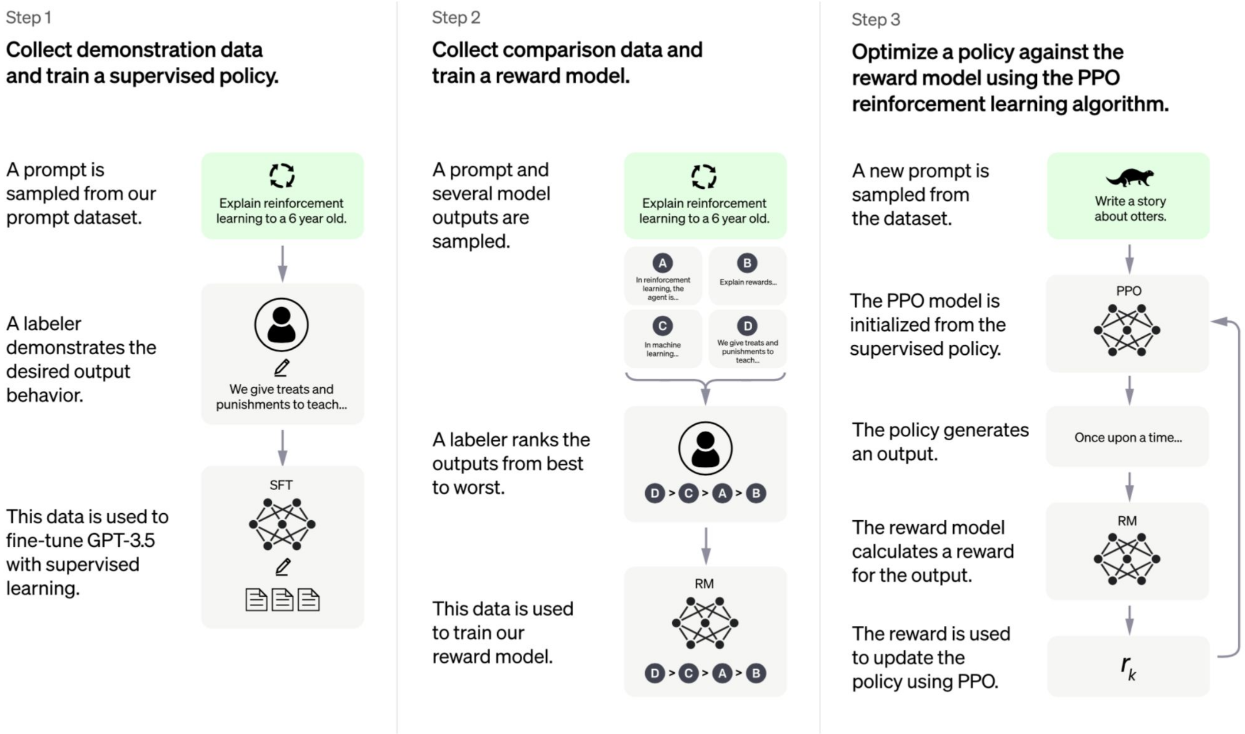
[Degrave et al. 2022]

Efficient and targeted COVID-19 border testing via RL



[Bastiani et al. 2021](#)

ChatGPT



openai.com

References

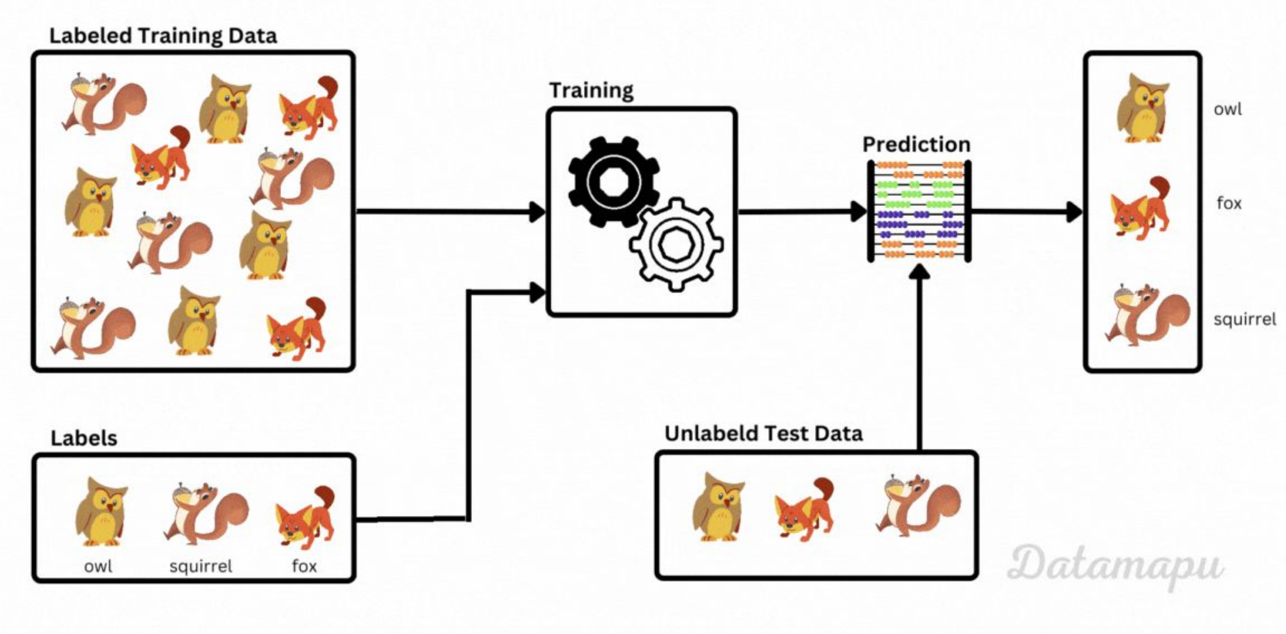
Courses at Stanford:

- CS 234 Reinforcement Learning
- CS 332 Advanced Survey of Reinforcement Learning
- MS&E 338 Reinforcement Learning

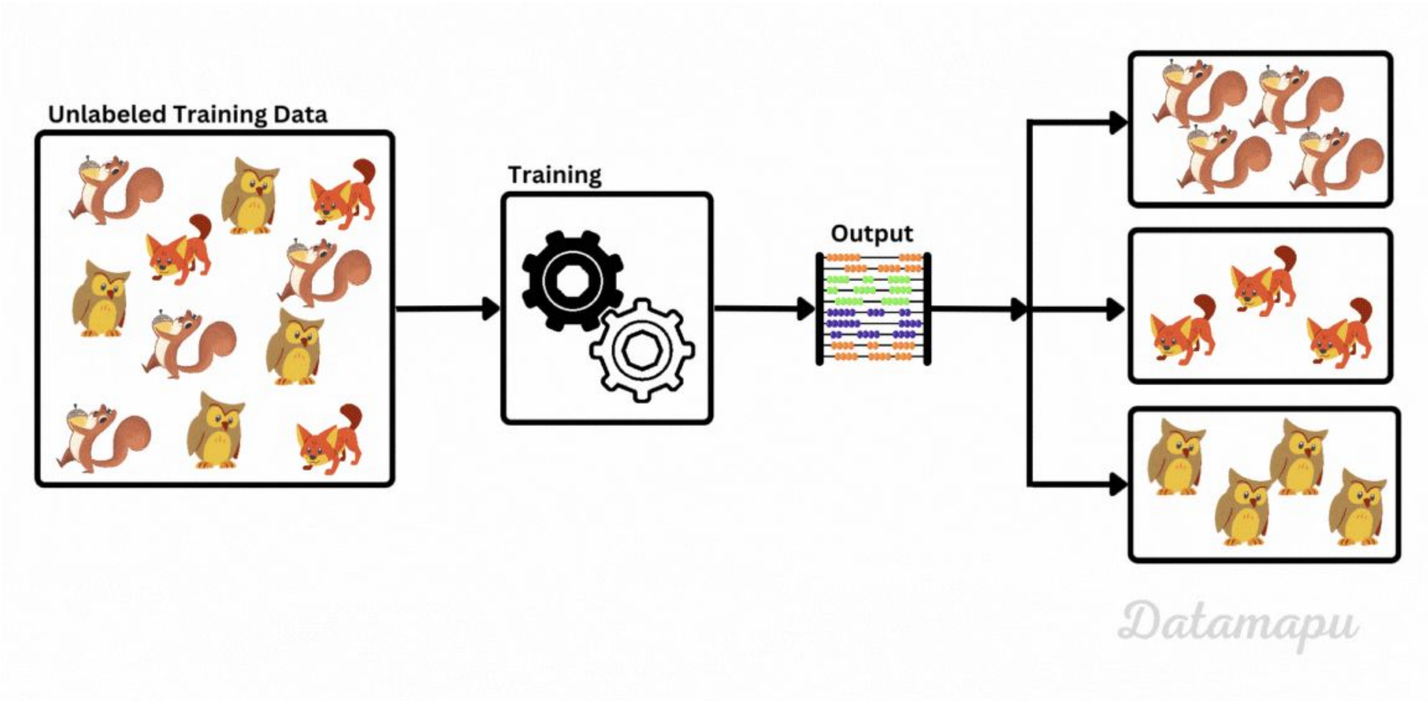
Other interesting websites:

- Mastering Reinforcement Learning
- Stable Baseline 3
- OpenAI Spinning Up

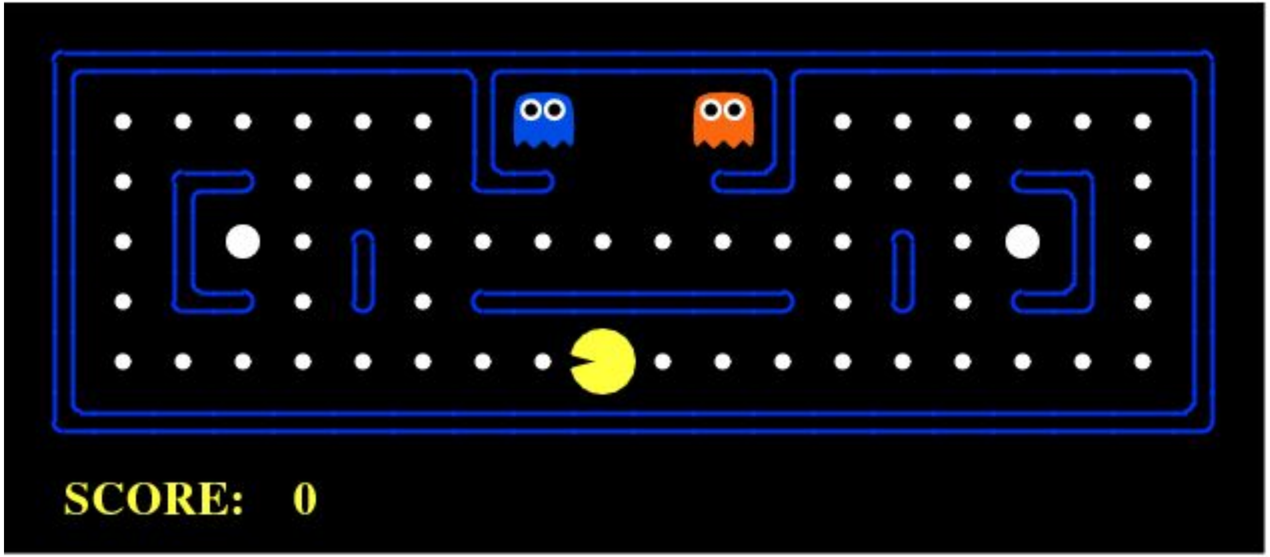
Machine Learning Paradigms



Machine Learning Paradigms



Machine Learning Paradigms



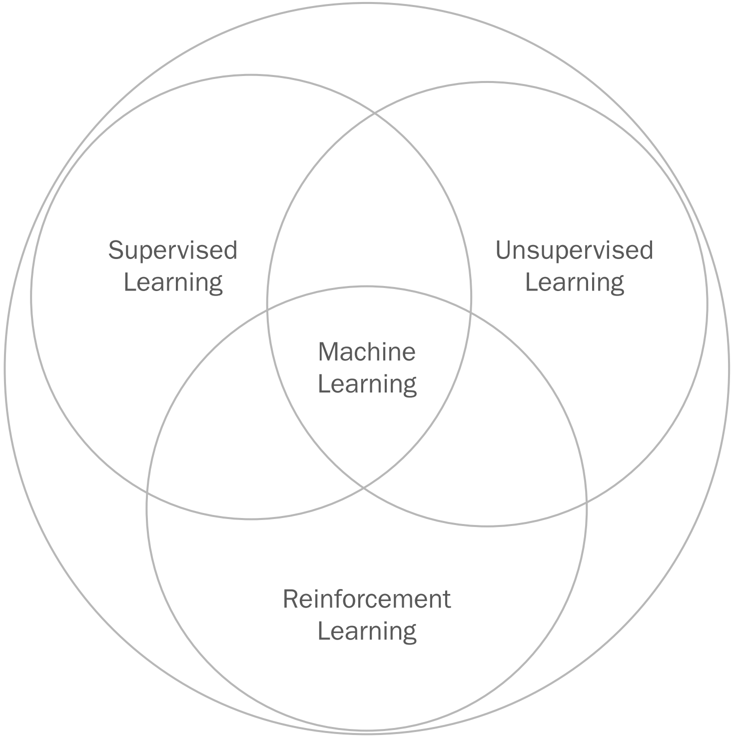
Learning Through Experience

A time-lapse of a baby playing with toys. Source.

- Learning through **experience/data** to make good decisions under uncertainty
- Essential part of intelligence
- Builds strongly from theory and ideas starting in the 1950s with Richard Bellman

Exploring Exploration: Comparing Children with RL Agents in Unified Environments. Eliza Kosoy, Jasmine Collins, David M. Chan, Sandy Huang, Deepak Pathak, Pulkit Agrawal, John Canny, Alison Gopnik, and Jessica B. Hamrick arXiv preprint arXiv:2005.02880 (2020)

Reinforcement Learning Framework

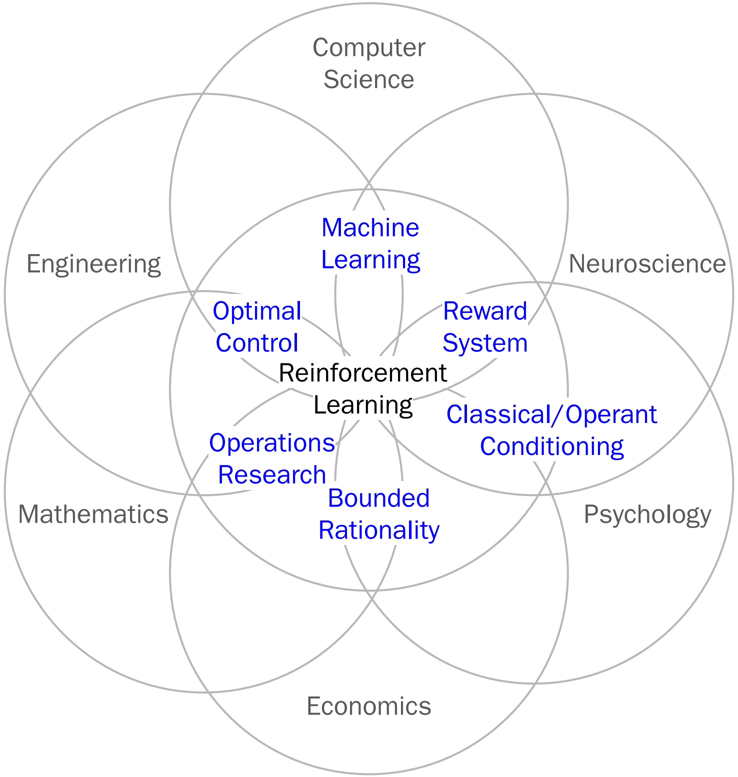


When we do reinforcement learning

- there is no supervisor, only a reward signal (e.g., this was good/bad, this gives you 10 points)
- Feedback may be delayed, but can also be instantaneous depending on the environment
- **Time really matters**: we talk about sequential processes (**non i.i.d data**)
- The agent is influenced by the sequence of data it receives

David Silver UCL-RL Course 2015

Reinforcement Learning Framework



Reinforcement learning is the science of decision-making → **try to find the optimal way to make decisions**

A number of impressive successes in the last decade

[David Silver UCL-RL Course 2015]

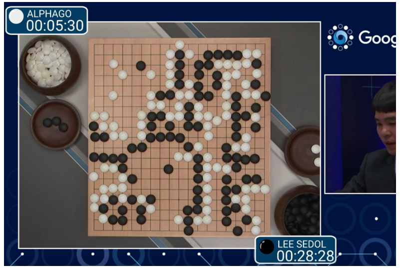
Examples

In human-robot interaction

- Understanding the human's need in a sequence of interactions → **maximise human's satisfaction**

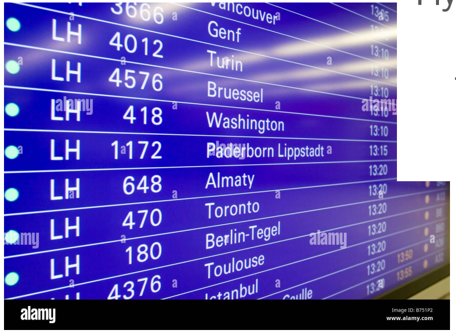
In package delivery in a city

- Figure out where to go and deliver what package → **maximise delivered packages**



AlphaGo

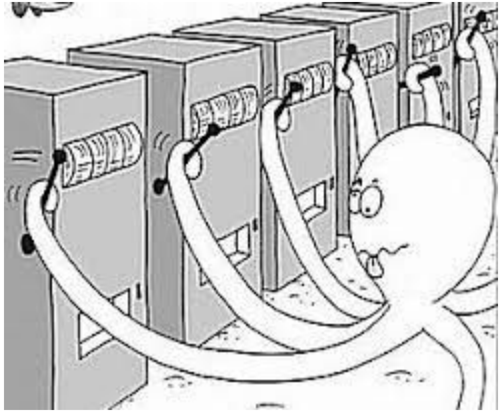
- Where to put the stones to beat the opponent → **maximise score**



Flying airplanes

- How to schedule your airplanes to go from one location to another → **maximise number of scheduled planes**

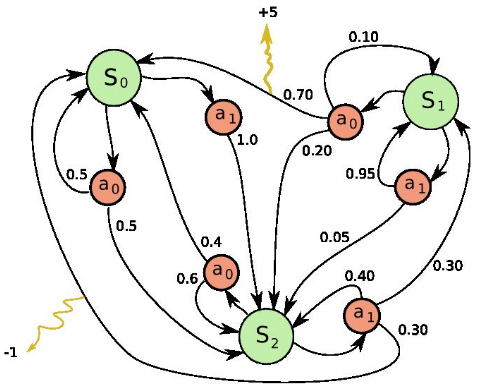
Sequentiality: Bandits problems vs Sequential problems



In **multi-armed bandit problems**, sequentiality is relatively simple:

- At each time step, you choose one action (pull one "arm")
- You immediately observe a reward for that action only
- The state of the world remains the same - arms don't change based on your previous choices
- Each decision is independent, though learning accumulates
- Your previous actions don't affect future reward distributions
- You see the consequence of your action right away
- The "state" is just your current belief about arm values
- The main challenge is balancing trying new arms vs. choosing known good ones

Sequentiality: Bandits problems vs Sequential problems



Sequential Decision Problems (MDPs/POMDPs)

- Goal: select actions to maximise total future reward
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Actions change the environment state
- Actions may have rewards that appear many steps later
- The sequence of actions matters, not just individual choices
- Hard to know which past actions led to current rewards

Basic decision-making problem (deterministic)

discrete-time optimal control problem → find the best sequence of control actions to minimize some cost while satisfying system dynamics and constraints

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Discrete-time model
Additive cost (central assumption)

[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \dots, N$

condition: state vector at time step k

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \dots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\dots,N-1} J(x_0; u_0, \dots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, \underline{u_k}), \quad k = 0, \ldots, N$

action: control input at time step k

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = \underline{f_k(x_k, u_k)}, \quad k = 0, \ldots, N$

State transition function (how the system evolves from one time step to the next)

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad \underline{k = 0, \ldots, N}$

discrete time steps and 0 to N (**finite horizon problem**)

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

The next state depends on the current state and the control we apply
No randomness; given state and control, next state is known

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $\underline{u_k \in U(x_k)}$

feasible control set that may depend on the current state

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

This represents physical limitations (e.g., maximum motor torque, speed limits)

The constraints can be state-dependent, meaning available actions may change based on where you are

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $\underline{J(x_0; u_0, \ldots, u_{N-1})} = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

total cost we want to minimize

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = \underline{g_N(x_N)} + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

terminal cost (penalty for where we end up)

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} \underline{g_k(x_k, u_k)}$

stage cost at each time step (running costs)

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

The cost combines: immediate costs at each step + final cost

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$\underbrace{J^*(x_0)}_{\text{Optimal value function (minimum achievable cost from initial state x\_0)}} = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; \underbrace{u_0, \ldots, u_{N-1}}_{\text{Control sequence}})$$

Basic decision-making problem (deterministic)

System: $x_{k+1} = f_k(x_k, u_k), \quad k = 0, \ldots, N$

Control constraints: $u_k \in U(x_k)$

Cost: $J(x_0; u_0, \ldots, u_{N-1}) = g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k)$

Decision-Making Problem:

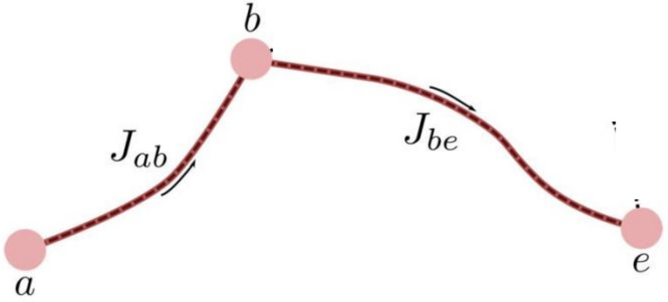
$$J^*(x_0) = \min_{u_k \in U(x_k), k=0,\ldots,N-1} J(x_0; u_0, \ldots, u_{N-1})$$

We're finding the control sequence that minimizes total cost. Subject to: system dynamics + control constraints

Principle of optimality (Bellman's Principle)

The key concept behind the dynamic programming approach is the principle of optimality

Suppose optimal path for a multi-stage decision-making problem is



- first decision yields $a - b$ segment with cost J_{ab}

- remaining decisions yield $b - e$ segments with cost J_{be}

- optimal cost is then $J^*_{ae} = J_{ab} + J_{be}$

[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Principle of optimality (Bellman's Principle)

Claim: If $a - b - e$ is optimal path from **a** to **b**, then $b - e$ is optimal path from **b** to **e**.

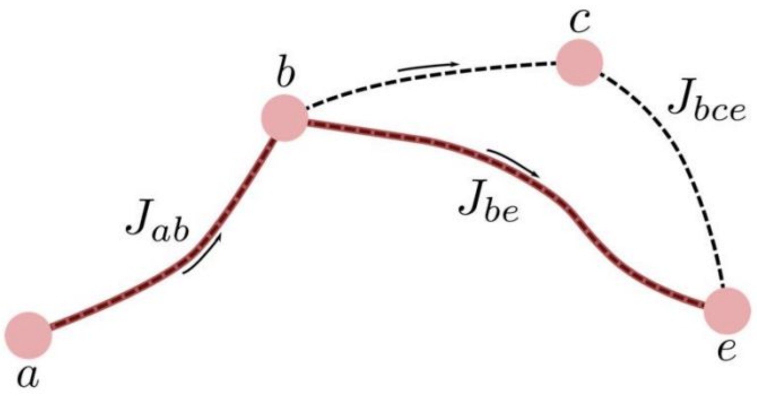
Proof: Suppose $b - c - e$ is the optimal path from **b** to **e**. Then $J_{bce} < J_{be}$

and

$$J_{ab} + J_{bce} < J_{ab} + J_{be} = J_{ae}^*$$

We found a path from a to e that's better than the "optimal" one

Contradiction!



[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Principle of optimality (Bellman's Principle)

Principle of optimality (for deterministic systems):

Let $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$

be an optimal control sequence, which together with x_0^*

determines the corresponding state sequence $\{x_0^*, x_1^*, \ldots, x_N^*\}$

Consider the subproblem whereby we are at x_k^*

at time k and we wish to minimize the cost-to-go from time k to time N, i. e.,

$$g_k(x_k^*, u_k) + \sum_{m=k+1}^{N-1} g_m(x_m, u_m) + g_N(x_N)$$

Then the truncated optimal sequence $\{u_0^*, u_1^*, \ldots, u_{N-1}^*\}$

is optimal for the subproblem

Tail of optimal sequences optimal for tail subproblems

[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Applying the principle of optimality

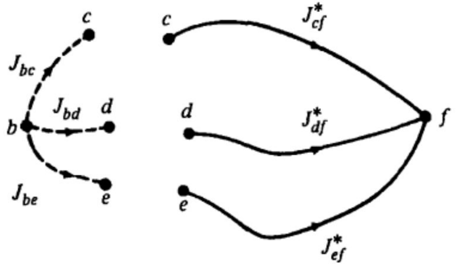
Principle of optimality: if $b - c$ is the initial segment of the optimal path from b to f, then $c - f$ is the terminal segment of this path

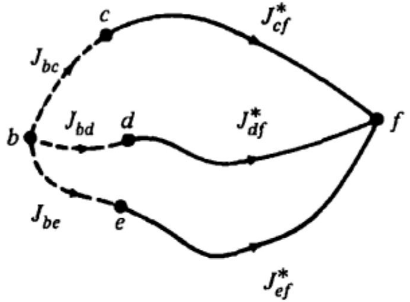
Hence, the optimal trajectory is found by comparing:

$$C_{bcf} = J_{bc} + J^*_{cf}$$

$$C_{bdf} = J_{bd} + J^*_{df}$$

$$C_{bef} = J_{be} + J^*_{ef}$$





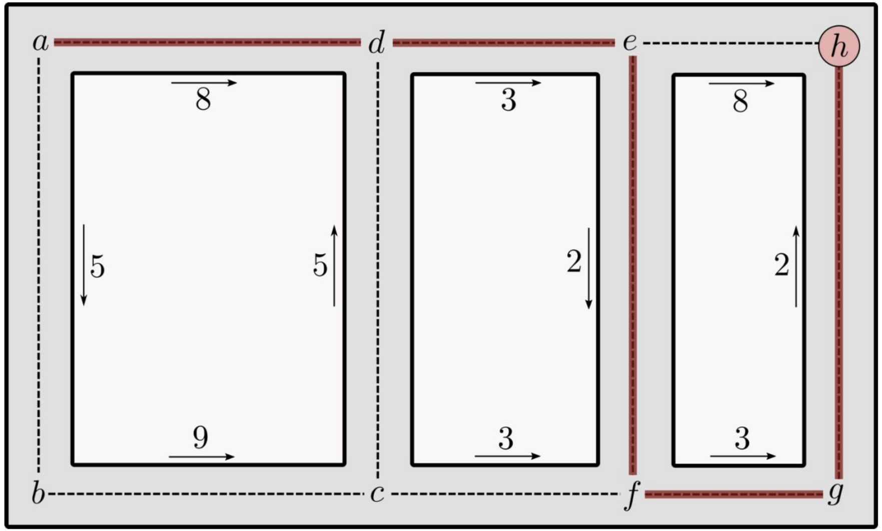
[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Applying the principle of optimality

- need only to compare the concatenations of immediate **decisions and optimal decisions** → significant decrease in computation / possibilities
- in practice: carry out this procedure <u>backward</u> in time

[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Example



Optimal cost: 18

Optimal path: $a \rightarrow d \rightarrow e \rightarrow f \rightarrow g \rightarrow h$

[Principles of Robot Autonomy II Markov decision processes and dynamic programming, Stanford and ASU 2025]

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for $k = N - 1, \dots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Once the functions $J_0^*, \dots, J_N^*$

have been determined, the optimal sequence
can be determined with a forward pass

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

backward pass: what's the optimal cost from any state?

and for $k = N - 1, \ldots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Once the functions $J_0^*, \ldots, J_N^*$

have been determined, the optimal sequence can be determined with a forward pass

Dynamic Programming Algorithm

Start at the **final time**
This sets the terminal cost as the boundary condition

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for $k = N - 1, \ldots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Once the functions $J_0^*, \ldots, J_N^*$

have been determined, the optimal sequence can be determined with a forward pass

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for $k = N - 1, \ldots, 0$, let

Work backwards through time

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Once the functions $J_0^*, \ldots, J_N^*$

have been determined, the optimal sequence
can be determined with a forward pass

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for k = N – 1, … , 0, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \Big[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \Big] \quad \text{for all } x_k$$

At each time step k, compute the optimal **cost-to-go function**

Once the functions $J_0^*, \ldots, J_N^*$

have been determined, the optimal sequence can be determined with a forward pass

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for $k = N - 1, \ldots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Bellman's principle: **the optimal cost from state x_k equals the minimum over all feasible controls of [immediate cost + optimal future cost].**

Once the functions $J_0^*, \ldots, J_N^*$

have been determined, the optimal sequence can be determined with a forward pass

Dynamic Programming Algorithm

Start with $J_N^*(x_N) = g_N(x_N), \quad \text{for all } x_N$

and for $k = N - 1, \ldots, 0$, let

$$J_k^*(x_k) = \min_{u_k \in U(x_k)} \left[g(x_k, u_k) + J_{k+1}^*(f(x_k, u_k)) \right] \quad \text{for all } x_k$$

Once the functions $J_0^*, \ldots, J_N^*$ have been determined, the optimal sequence can be determined with a forward pass

After computing all value functions you construct the actual optimal trajectory

Dynamic Programming Algorithm

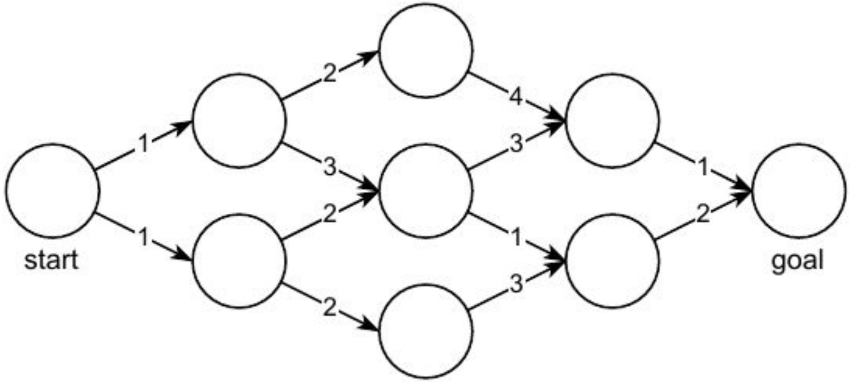
Forward Pass

- Start with given initial state x_0
- At each time k, find the control that achieves the minimum in the Bellman equation
- Apply that control to get the next state
- Repeat until reaching the final time

what should I actually do?

Dynamic Programming Algorithm

- DP guarantees **finding the globally optimal solution** (not just locally optimal) due to the **principle of optimality**
- The algorithm must be computed for all possible states x_k, which can be computationally challenging in high-dimensional problems.



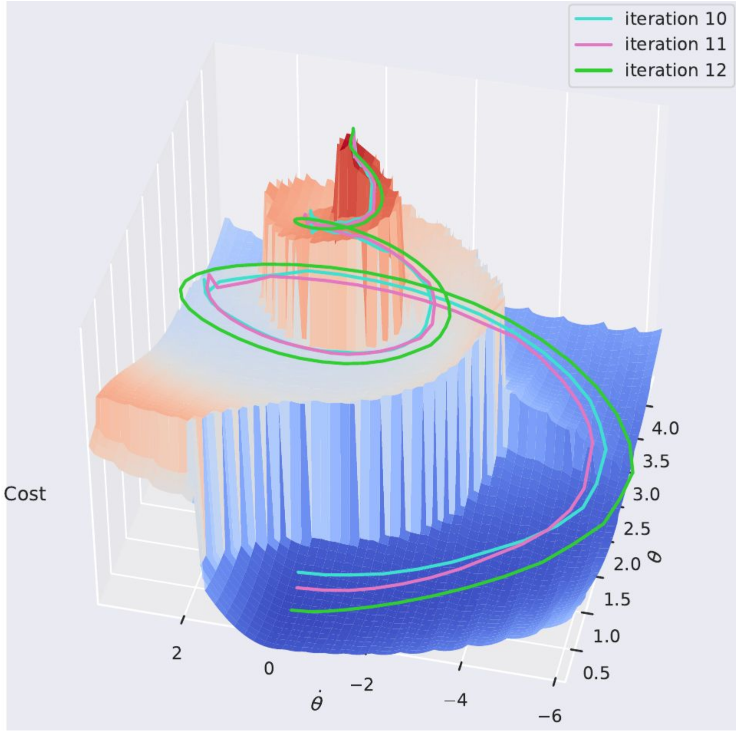
[redacted]

The approach that is neither dynamic nor involves programming. Thanks Dr Bellman!

imgeorgiev.com

Dynamic Programming Algorithm

- The algorithm must be computed for all possible states x_k, which can be computationally challenging in high-dimensional problems.
    - Instead of computing the values over the entire state space via DP, **we can be smarter by only computing the values around our best-guess trajectory** and **iteratively closing in on the actual optimal solution.**



imgeorgiev.com

Rewards

- Scalar feedback R_t about how well the agent is doing at time step t
- The goal of the agent is to maximise the cumulative reward
- Reinforcement learning is based on the **reward hypothesis**:

All goals can be expressed by the maximisation of expected cumulative rewards

First step is **understanding the reward signal**

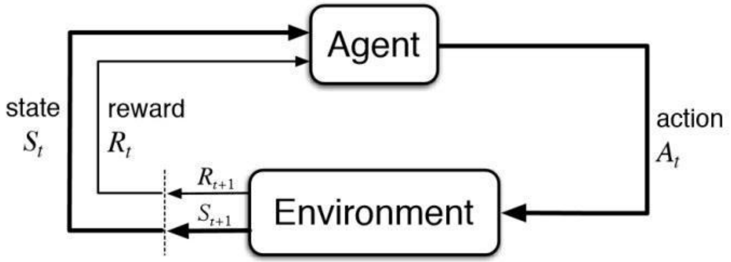
Exploration/exploitation tradeoff

To figure out what is the meaning of the scalar value the agent’s has to explore

Exploration/exploitation tradeoff

- Should the agent explores what it knows or looks for new solutions?
- How fast should you decrease your exploration rate?
- Epsilon-greedy: taking the best action most of the time and a random action from time to time

Terminology



Reinforcement Learning: An Introduction,
[Richard S. Sutton and Andrew G. Barto (2018)]

- A learning agent has to:
  - Sense the state of its environment
  - Take actions that affect the state
  - Have a goal or goals relating to the state of the environment

A Markov Decision Process (**MDP**) is a mathematical formalisation for modeling decision making and include these aspects:

- Sensation
- Action
- Goal

Terminology

- **Policy** - maps from perceived states of the environment to actions to be taken in those states (e.g., in psychology stimulus-response rules or associations)
- **Reward signal** - defines the goal in the problem. Single number that tells what are the good and bad events for the agent (e.g., in biological systems pleasure/pain)
- **Value Function** - specifies what is good in the long run
- **Model of the environment** - mimics the behavior of the environment

Offline vs Online Planning

Offline planning (MDPs)

- MDP is given
- The agent find the optimal policy for the MDP
- The agent acts in the environment

Online Planning (RL)

- Learning is required
- The agent has access to a set of available actions and information about the state it is in.

Solving MDPs

Techniques for solving MDPs (and POMDPs) can be separated into three categories:

- **Value-based techniques** aim to learn the value of states (or learn an estimate for value of states) and actions: that is, they learn value functions or Q functions. We then use policy extraction to get a policy for deciding actions.

- **Policy-based techniques** learn a policy directly, which completely by-passes learning values of states or actions all together. This is important if for example, the state space or the action space are massive or infinite. If the action space is infinite, then using policy extraction is not possible because we must iterate over all actions to find the optimal one. If we learn the policy directly, we do not need this.

- **Hybrid techniques that combine value- and policy-based techniques.**

Value Iteration

Reinforcement Learning: An Introduction,
[Richard S. Sutton and Andrew G. Barto (2018)]

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

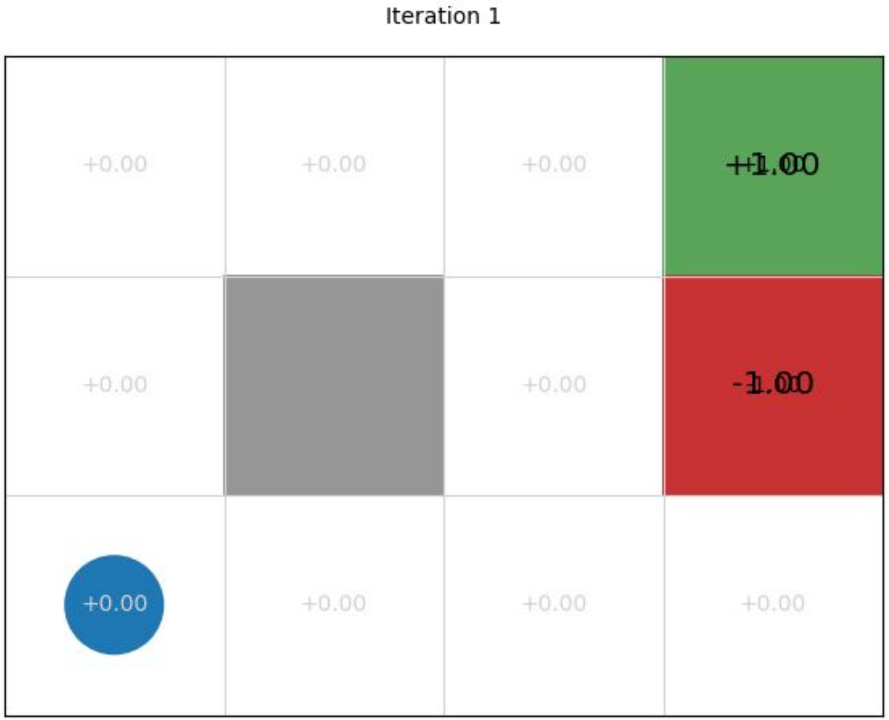
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

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

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

Value Iteration

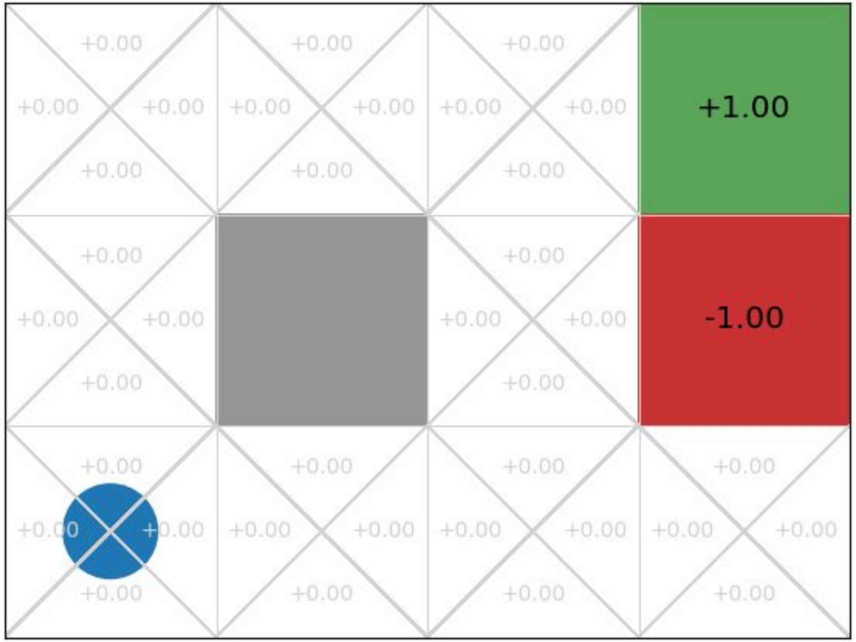
[Value Iteration, University of Queensland]



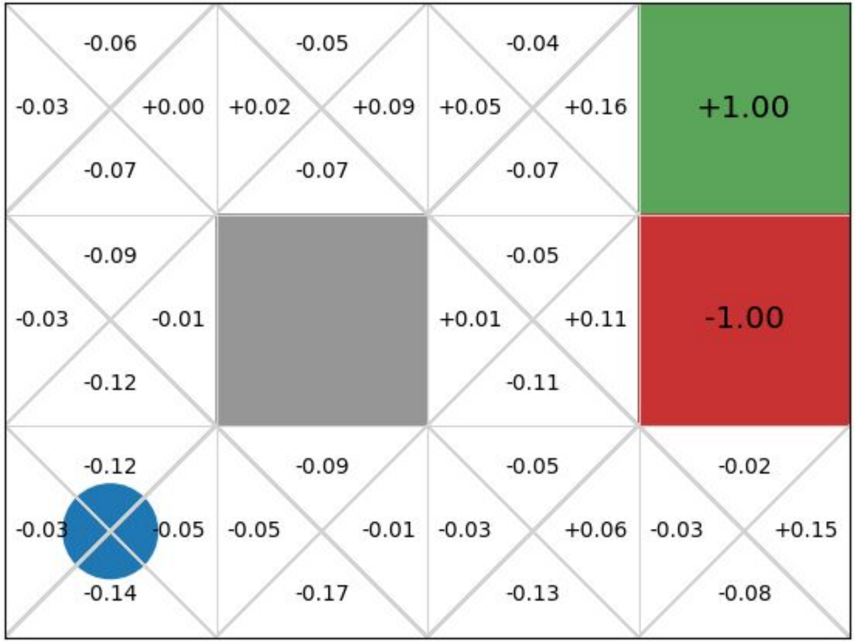
Q-learning vs Deep Q-learning

gibberblot.github.io/rl-notes/single-agent/function-approximation.html

Linear Q-learning after episode 0



Deep Q-learning after episode 0



Policy Iteration

Reinforcement Learning: An Introduction,
[Richard S. Sutton and Andrew G. Barto (2018)]

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 $s \in \mathcal{S}$

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

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

Value Iteration vs Policy Iteration

stackoverflow.com

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 $s \in \mathcal{S}$:
           $temp \leftarrow v(s)$
           $v(s) \leftarrow \sum_{s'} p(s'|s,\pi(s))\Big[r(s,\pi(s),s') + \gamma v(s')\Big]$
           $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
   until $\Delta < \theta$ (a small positive number)

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

Figure 4.3: Policy iteration (using iterative policy evaluation) for v_*. This algorithm has a subtle bug, in that it may never terminate if the policy continually switches between two or more policies that are equally good. The bug can be fixed by adding additional flags, but it makes the pseudocode so ugly that it is not worth it. :-)

finding optimal value function

Initialize array v arbitrarily (e.g., $v(s) = 0$ for all $s \in \mathcal{S}^+$)

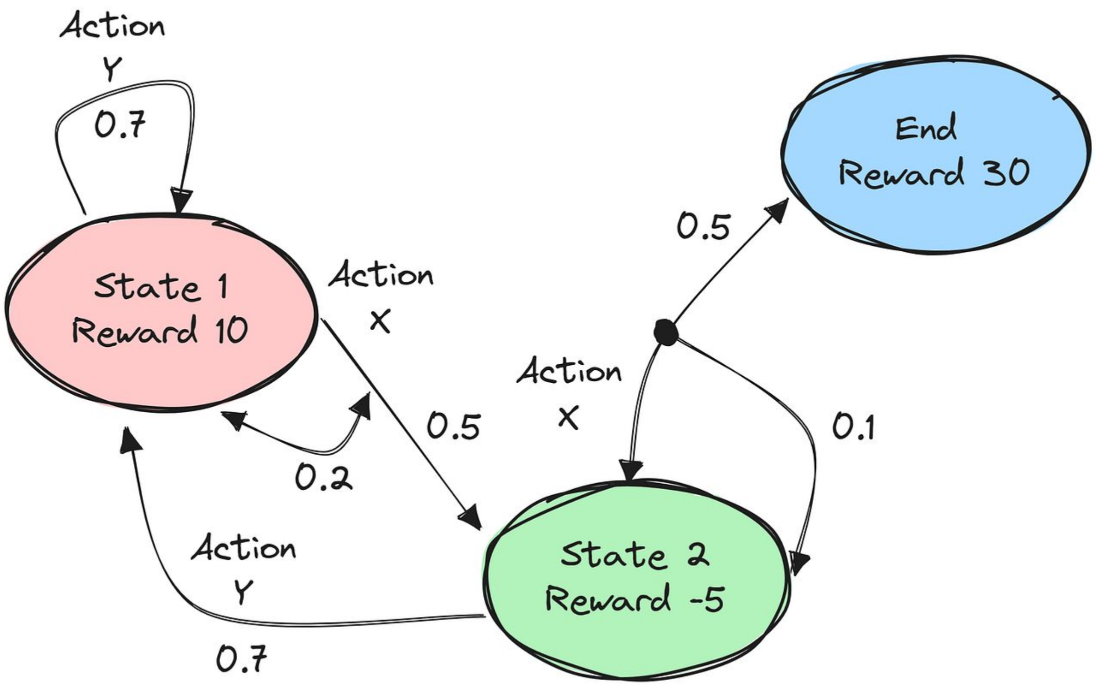
Repeat
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $temp \leftarrow v(s)$
        $v(s) \leftarrow \max_a \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma v(s')]$
        $\Delta \leftarrow \max(\Delta, |temp - v(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
    $\pi(s) = \arg\max_a \sum_{s'} p(s'|s,a)\Big[r(s,a,s') + \gamma v(s')\Big]$

Figure 4.5: Value iteration.

one policy update (extract policy from the optimal value function

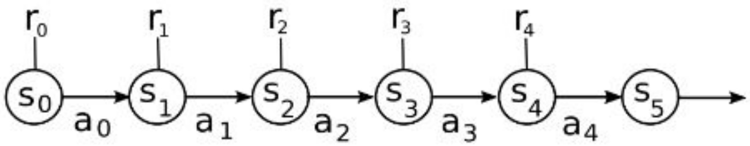
Stochastic transition function



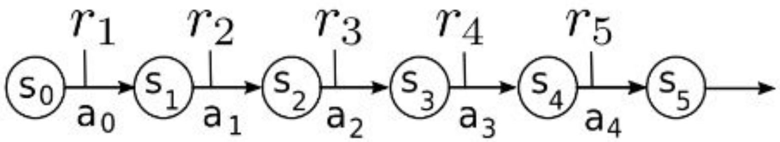
python.plainenglish.io

Rewards over states or actions

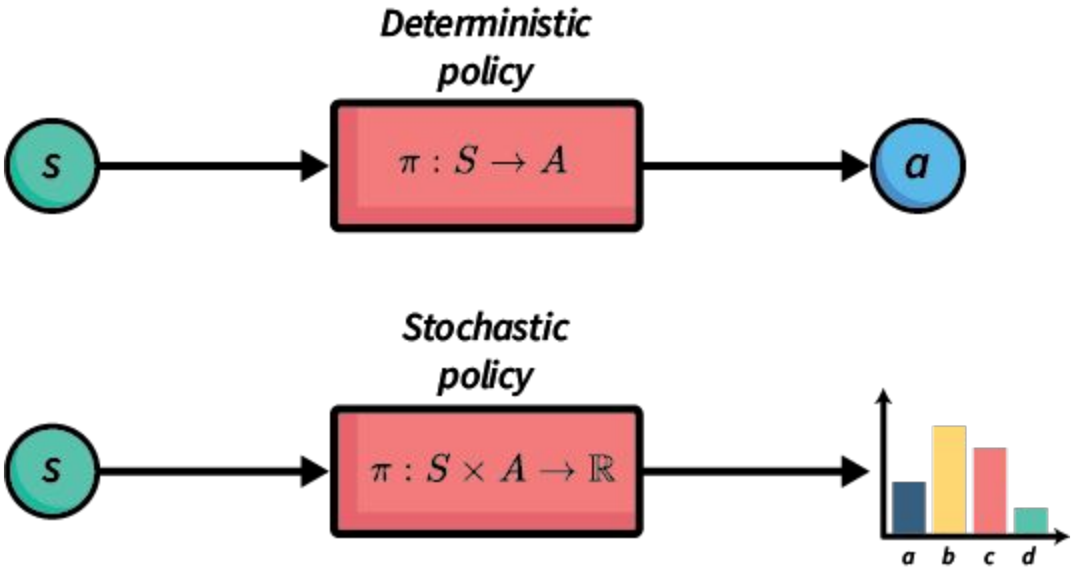
Rewards over states



Rewards over actions in states



Deterministic versus stochastic policy



Model-based Vs Model-Free RL

Model-based RL:

- the agent learns an explicit model of the environment, including the transition function and reward function.
- Components:
  - Transition model: P(s' | s, a) - probability of next state given current state and action
  - Reward model: R(s, a, s') - expected reward for taking action a in state s and ending up in state s'

Model-free RL:

- learns to make decisions directly from experience, without explicitly modeling the environment.
- **Value-based**:
  - Learn value functions (e.g., Q-learning, SARSA)
- **Policy-based**:
  - Learn policy directly (e.g., REINFORCE)
- **Actor-Critic**:
  - Combine value and policy learning

On-policy vs Off-policy learning

on-policy learning

the agent learns about and improves the policy that it's currently following

- It learns from actions actually taken by the current policy

off-policy learning

- the agent learns about a target policy different from the one it's following to collect experiences.
- Uses a behavior policy to collect experiences and a target policy that is being learned and improved
- Can learn from historical data or experiences generated by other policies.

Cool Assignment

CS 234 Winter 2025 Assignment 1

Q-learning and SARSA

[Q-Learning and SARSA]()

Reinforcement Learning Tutorial (with Open AI Gym)

[Reinforcement Learning with Gym Envs]

HUMANE AI NET

SORBONNE UNIVERSITÉ

VU UNIVERSITY AMSTERDAM