Introduction to Reinforcement Learning

Aprendizaje Automático del máster *Ingeniería del Software e Inteligencia Artificial*

Jesse Read

Version: December 10, 2024

Plan for Today

1. Objectives and Context
2. Introduction: Motivation and Applications
3. General Setup: Agent and Environment; State, Action, Reward
4. Stochastic Environments and MDPs
5. Learning: Return (Gain) and Value
6. Questions and Discussion

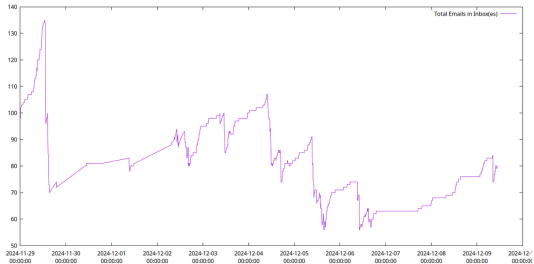
Objectives and Context

This Lecture: Objectives

- What is Reinforcement Learning (RL) in context of this talk (vs search, optimization, supervised learning, ...)
- Why RL (potential applications, and motivation)
- How to set up a RL problem ← probably the most important part
  We will focus on a few toy examples
- Main concepts and fundamentals, up until and including value functions
- Solving a toy problem with Monte Carlo RL
- Practical challenges of RL
- Convince you that with what we covered so far, we can scale up to real-world problems, by plugging in appropriate deep-learning architectures
- If there's time: discuss different algorithms, and a more detailed positioning of RL in the context of modern AI, including open research in this area
- (More) questions

Types of Models (Where we might use Machine Learning)

- **Descriptive models** (data mining, pattern mining, data analytics, . . .)
  $\Rightarrow$ describe the past and present
- **Predictive models** (machine learning, statistical methods, extrapolation, forecasting, . . .)
  $\Rightarrow$ use the past and present to predict the future
- **Prescriptive models** (optimisation and autonomous agents, reinforcement learning, solvers, planning, . . .) $\leftarrow$ In this course, we are here
  $\Rightarrow$ take actions that will change the future



Progress in Context

2004:

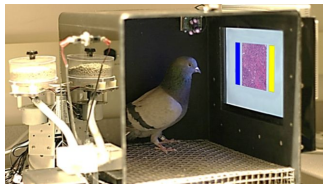
- Human can learn how to play Go in a few minutes and go on to beat the state-of-the-art AI
- DARPA Grand Challenge: None of the autonomous vehicles finished; the best team completed 11.78 of the 240 km
- Computer vision still largely unsolved. Focus on mining text associated with any given image, or 'bag of pixels'.
- Speech recognition had plateaued, needed heavy personalized training; Machine translation provided 'amusing' examples

2024, Already seems like old news:

- AI has beaten world champion in Go
- Autonomous vehicles drive hundreds of kilometres through urban environments
- Computer vision and speech recognition penetrating the mass market (Siri, Google Translate, etc.);
  Impressive results from of language models, text-to-image models (ChatGPT, Midjourney, ...).

But still a long way to go. Consider AI vs Pigeon:

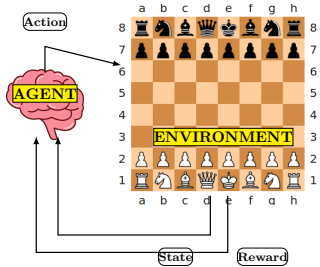
- Getting from point A to B without crashing into anything, over long distances, in 3d coordinates, and orders of magnitude less power than modern AI
- Impressive visual skills: **Pigeons identify breast cancer 'as well as humans'**: Pigeons, with training, did just as well as humans in a study testing their ability to distinguish cancerous from healthy breast tissue samples. [...] Likely no bigger than the tip of your index finger, the pigeon's brain nonetheless has impressive capabilities ... Pigeons can distinguish identities and emotional expressions on human faces, letters of the alphabet, misshapen pharmaceutical capsules, and even paintings by Monet vs Picasso.



– BBC (2015); Scientific article: *Levenson et al., Pigeons (Columba livia) as Trainable Observers of Pathology and Radiology Breast Cancer Images*, PLOS ONE 2015; Turner and Edward *The pigeon as a machine: Complex category structures can be acquired by a simple associative model.* Iscience 26.10 (2023).

Introduction: Reinforcement Learning

- No data set; only an environment
- No training labels; only reward signal
- The model is an agent; map state-observations to actions
- Sequential decision making (decisions affect the future ← **huge implications!!!**)
- Main challenge: associating agent actions with rewards through time



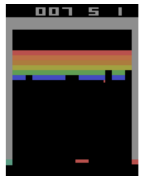
Reinforcement Learning is a means to obtain an intelligent agent[1].

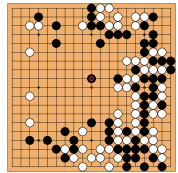
[1] or, autonomous agent, rational agent, an 'AI'

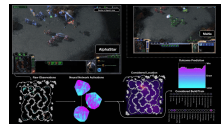
Introduction: Motivation and Applications

Applications: Games


(a) Chess


(b) Atari


(c) Go

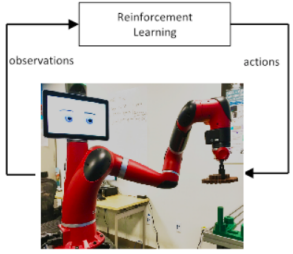

(d) StarCraft II


(e) Dota

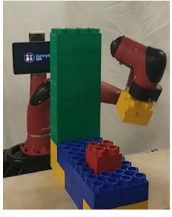
- 1997 Deep Blue (search-based) beats world champion at Chess
- 2013 DeepMind (DQL) surpasses human expert on some Atari 2600 games
- 2016 DeepMind AlphaGo (MCTS) beat world champion
- 2019 DeepMind AlphaStar (PG, IL, . . .) acquires grandmaster status
- 2019 OpenAI agents (PPO) defeats world champion team in live Dota match

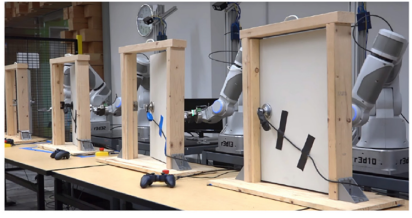
▸ FPS-Quake ▸ Starcraft-AlphaStar

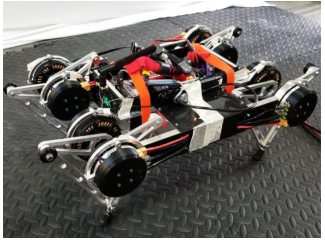
Applications: Robotics and Autonomous Vehicles



▸ Helicopter ▸ Robot ▸ More Robots



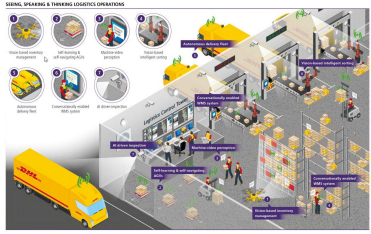




Ibarz et al., *How to train your robot with deep reinforcement learning: lessons we have learned*, 2021

Applications: Logistics, Finance and Business

- Trading/finance/manage investment portfolio ▸ IBM
- In recommendation engines, email advertising
- Marketing and advertising; real-time bidding ▸ Advertising/Bidding ▸ Bidding

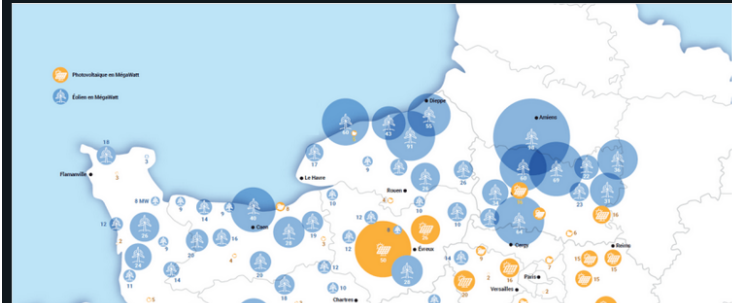


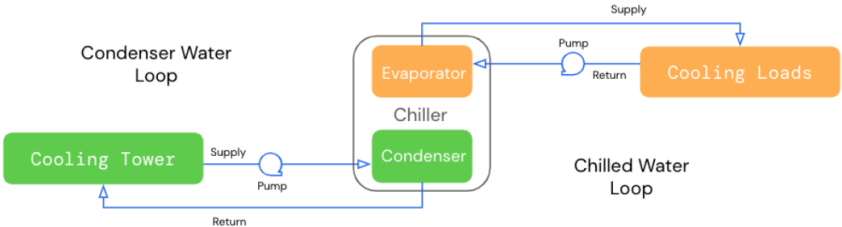
Application: Energy Systems

La Région Ile-de-France et RTE lancent le Challenge IA pour la Transition Energétique

18.05.2023 • TRANSITION ÉNERGÉTIQUE

Pour assurer l'alimentation électrique du territoire à chaque seconde, RTE surveille et pilote les flux d'électricité 24 heures sur 24, 7 jours sur 7, 365 jours par an. Le développement des énergies renouvelables, indispensable à la transition énergétique, nécessite d'adapter la gestion du système électrique. En effet, ces énergies sont variables en fonction des conditions météorologiques et réparties sur tout le territoire. Cela implique un pilotage en temps réel encore plus performant, avec des données plus nombreuses à traiter pour anticiper au mieux les situations. L'Intelligence Artificielle est une solution pour accompagner les dispatchers de RTE dans leurs missions.





Luo et al., Controlling Commercial Cooling Systems Using Reinforcement Learning, 2022. N.B. Also ▸ Energy

Technology

Software update for world's wind farms could power millions more homes

An AI that predicts wind changes could boost wind turbine efficiency by 0.3 per cent, which globally would amount to enough extra electricity to keep a country running

By Matthew Sparkes

21 May 2023

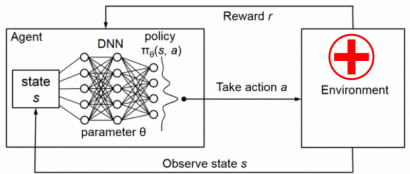
A small efficiency gain for every wind turbine would lead to a big boost globally

Justin Paget/Digital Vision/Getty Images

Current algorithms track wind patterns and adjust the turbine blades to anticipate changes, but Alban Puech and Jesse Read at the Polytechnic Institute of Paris believe artificial intelligence can do better. They have trained a reinforcement-learning algorithm to monitor wind patterns and develop its own strategy to maintain the turbine at the correct angle. And because

Applications: Education, Healthcare, Agriculture, Politics, LLMs ...

- Healthcare: Diagnosis, manage hospital resources, ... ▸ Health
- Education: Personalised/auto-generated curriculum, ...
- Farming and Agriculture: Managing resources, ...
- Sports
- Politics: Obtaining a policy ▸ Politics

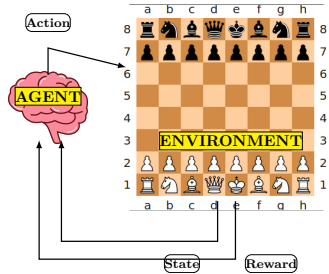


- Large Language Models (and other deep neural networks)

General Setup: Agent and Environment; State, Action, Reward

General Setup for Reinforcement Learning: Agent and the Environment

1. The **agent** observes the state of the **environment**, and receives a reward
2. The agent takes an available action
3. The next state is determined by environment dynamics ('rules of the game')



Important: The agent may alter the environment (and thus its own future state observations *and reward*!)

Environment-agent interaction (simplified):

```
env = Environment()
agent = Agent(env)

while True:
    a = agent.act(s)
    s_next, r = env.step(a)
    agent.update((s,a,r,s_next))
    s = s_next
```

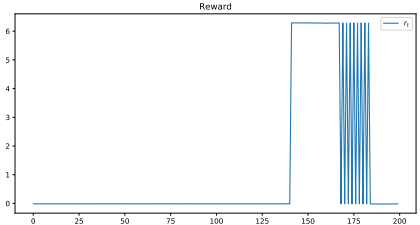
The Reward Signal and Trajectories

At time t we observe s_t, take action a_t, obtain reward r_{t+1}, and advance to next state s_{t+1}. An episode (or trajectory, τ):

$$\tau = \{s_1, a_1, \quad r_2, s_2, a_2, \quad r_3, s_3, a_3, \quad \ldots, \quad r_{T-1}, s_{T-1}, a_{T-1}, \quad , r_T, s_T\}$$

Challenges:

- Sparse rewards, e.g., $r_t = 0$ most of the time
- Weak rewards (limited impact of actions on reward)
- Temporal credit assignment: Which actions (a_{t-1}? a_{t-100}? both? none?) led to reward r_t?

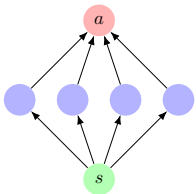


Agent and Policy

The policy defines the behaviour of the agent;

$$a_t = \pi(s_t)$$

indicates the action to take given the observation of the current state; i.e., the agent observes s_t and takes action a_t in response.



Neural Network

State s_t	Action a_t
0	$+1$
1	$+1$
2	-1
3	$+1$

Q-Table

```
def act(s):
    if s < 3:
        # Go Right
        return 1
    else:
        # Go Left
        return -1
```

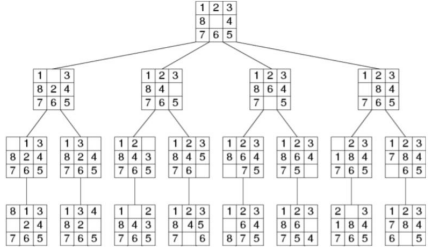
'Hardwired' policy

We can also have a stochastic policy $P(A_t \mid S_t = s_t) = \pi(s_t)$.

Search (vs RL)

It's just a search (each node is a state):

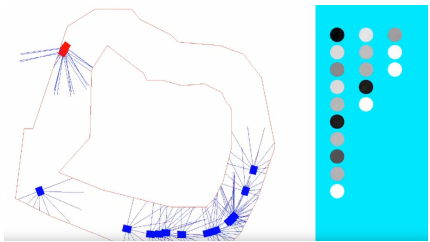
1. Generate the search tree
2. Collect sum of rewards at leaf
3. Backup
4. Choose the best branch



Does not scale!

Optimization (vs RL)

1. Generate a policy (or several)
2. Evaluate (check sum of rewards obtained)
3. Choose the best; [modify and] repeat



source: Youtube

Does not scale!

Imitation Learning (vs RL)

1. Observe an expert take action a_t from state s_t
2. Collect a dataset for $t = 1, 2, \ldots$
3. Train your favourite classifier (or regressor)
4. Update as necessary

a Kinesthetic teaching **b** Teleoperation teaching

Image source: Fang et al. *Survey of imitation learning for robotic manipulation*. 2019.

Imitation Learning is Supervised Learning, *not* RL.

Scales quite well; but usually **does not work** (OK, can say the same about RL on many problems, but . . .) and is **fundamentally limited** – cannot generalise

Stochastic Environments and MDPs

Stochastic Environments

Deterministic environments: if agent chooses to `step forward` (action), the agent move forwards (next state). **Stochastic environments**, e.g., real world:

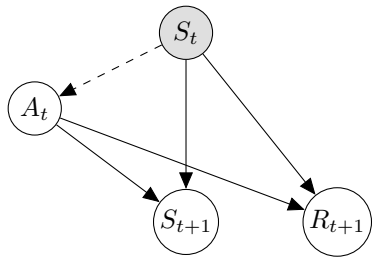
Image credits: unknown/can't remember

The agent takes action a from state s and arrives to state s' **with probability** $s' \sim p(\cdot \mid s, a)$. A stochastic adversary makes your environment stochastic too!

A Markov Decision Process (MDP): Model of the Environment

In stochastic environments, we cannot choose the next state deterministically.

- $\mathcal{S}$ state space, $s \in \mathcal{S}$
- $\mathcal{A}$ action space, $a \in \mathcal{A}(s)$
- $\mathcal{R}$ reward space, $r(s, a) \in \mathcal{R}$
- $r(s, a)$ reward function
- $p(s' \mid s, a)$ transition function (dynamics)
- $\pi(a \mid s)$ policy of the agent



In Reinforcement Learning we do not necessarily know this model.

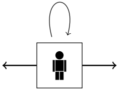
Slippery Floor Environment

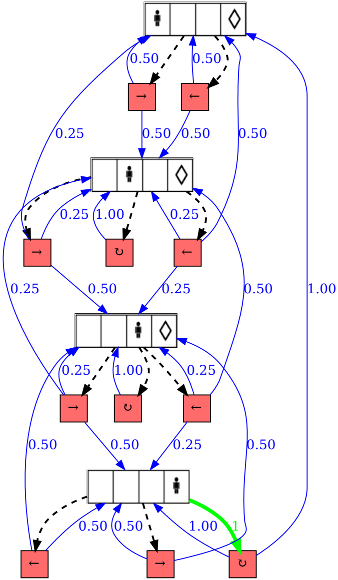
- state $s \in \{1, 2, 3, 4\}$
- action $a \in \{\leftarrow, \circlearrowright, \rightarrow\}$
- reward function $r(4, \circlearrowright) = \mathbf{1}$
  else $r(s, a) = 0$ for other values of s, a
- transition function $s' \sim p(\cdot \mid s, a)$

e.g.,

$p($  $\mid$  $, \leftarrow) = 0.5$

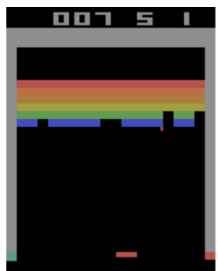
The floor is slippery!! And the lights are off, agent only sees 's':





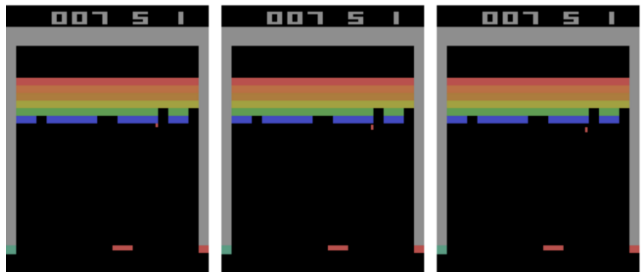
(*transition graph* representing MDP dynamics)

Markov Decision Process:
Does your environment satisfy the Markov property?
(Is it possible to perform optimally from any state s_t?)

$s_t =$ 

Important consideration! Much of Reinforcement Learning is built around the Markov assumption!

Markovian state:

$s_t =$ 

Other solutions: LSTMs, track the difference across frames, ...

i.e., if your decision process is not Markov, you can make it one (not necessarily easily)!

Learning: Return (Gain) and Value

Reinforcement Learning: Produce a Policy for an Environment

The output of reinforcement learning is a policy

$$a_t = \pi(s_t)$$

(for any state $s_t \in \mathcal{S}$) – but *what* policy?

Reinforcement Learning: Produce a Policy for an Environment

The output of reinforcement learning is a policy

$$a_t = \pi(s_t)$$

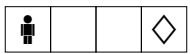
(for any state $s_t \in \mathcal{S}$) – but *what* policy?

The best policy should take the best action from the current state;

$$a_t^* = \pi(s_t)$$

i.e., the action to to optimize (**this is the important question**!) ??? ...

to optimize what?

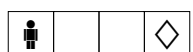


The Gain (Finite Scenario)

The gain (aka return[2] or sum of future rewards) at step t is

$$\begin{aligned} G_t &= \sum_{i=t}^{T} R_{i+1} \\ &= R_{t+1} + R_{t+2} + \ldots + R_T \end{aligned}$$

i.e., the sum of rewards of the episode; it indicates the value at current time t.



[2]We call it G the gain to avoid confusion with R the reward

The Gain (Finite Scenario)

The gain (aka return[2] or sum of future rewards) at step t is

$$\begin{aligned} G_t &= \sum_{i=t}^{T} R_{i+1} \\ &= R_{t+1} + R_{t+2} + \ldots + R_T \end{aligned}$$

i.e., the sum of rewards of the episode; it indicates the value at current time t.

(green for gain). But: when $T = \infty$? And $r_{t+1} = 1$ vs $r_{t+1000} = 1$?

[2] We call it G the gain to avoid confusion with R the reward

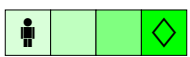
The Return/Gain (Infinite Scenario)

The return (aka gain) at step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

with discount factor $\gamma \in (0, 1)$ which indicates the relative value of closer rewards.



Gain G from each possible s of optimal agent

Task: The agent should take actions A_t to maximize G_t! The policy is implicit!

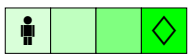
The Return/Gain (Infinite Scenario)

The return (aka gain) at step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

with discount factor $\gamma \in (0, 1)$ which indicates the relative value of closer rewards.



Gain G from each possible s of optimal agent

Task: The agent should take actions A_t to maximize G_t! The policy is implicit!

But

- G_t is from the future! The future is uncertain!
- G_t inherits the randomness (uncertainty) from environment *and* agent!

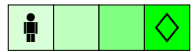
The Return/Gain (Infinite Scenario)

The return (aka gain) at step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

with discount factor $\gamma \in (0, 1)$ which indicates the relative value of closer rewards.



Gain G from each possible s of optimal agent

Task: The agent should take actions A_t to maximize G_t! The policy is implicit!

But

- G_t is from the future! The future is uncertain!
- G_t inherits the randomness (uncertainty) from environment *and* agent!

Therefore: We should maximize expected gain $\mathbb{E}[G_t]$, also called *value*.

Value = Expected Gain

Reinforcement learning is about maximising value:

$$\pi_* = \max_{\pi} \mathbb{E}[G_t]$$
$$a_t^* = \pi_*(s_t)$$

= minimising expected loss = the general setting of machine learning!

Remarks:

- gain G_t calculated according to discount factor γ and from time t
- expectation (randomenss) $\mathbb{E}$ from our policy π and the environment dynamics p

The Value Function

The value function (aka state-value function),

$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s]$$

maps a state to a value; which is the expected return from that state following policy π interacting with the environment.

We may think of a vector of $|\mathcal{S}|$ values; if $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$:

	s_1	s_2	s_3	s_4
V				

A V-function and Policy for the FROZEN LAKE Environment:

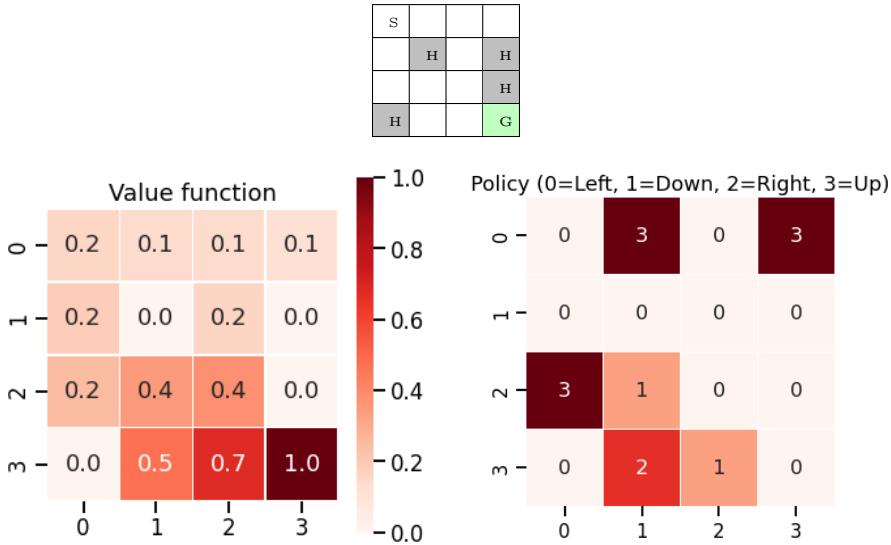


Image credits: Jérémie Decock

The Action-Value Function

The action-value function,

$$Q^{\pi}(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a]$$

maps a state *and action* to a value.

We may think of a table of $|\mathcal{S}| \times |\mathcal{A}|$ values; with $\mathcal{A} = \{a_1, a_2\}$:

Q	a_1	a_2
s_1		
s_2		
s_3		
s_4		

Monte Carlo RL

Goal: Find $V^\pi(s) = \mathbb{E}[G_t \mid S_t = s]$ for all s. Then just change s for more value!

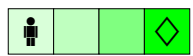
Problem (1): We don't have $V^\pi(s)$, we have to calculate (learn) it!

Problem (2): The expectation is annoying!

Solution: Monte Carlo Approximation!

$$V^\pi(s) = \mathbb{E}[G_t \mid S_t = s] \approx \frac{1}{N} \sum_{n=1}^{N} g_n^{(s)}$$

for N trajectories, ending in gain g_n; i.e., play the game N times, fill the value table,



then deploy the implicit policy.

Monte Carlo RL

Goal: Find $V^\pi(s) = \mathbb{E}[G_t \mid S_t = s]$ for all s. Then just change s for more value!

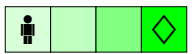
Problem (1): We don't have $V^\pi(s)$, we have to calculate (learn) it!

Problem (2): The expectation is annoying!

Solution: Monte Carlo Approximation!

$$V^\pi(s) = \mathbb{E}[G_t \mid S_t = s] \approx \frac{1}{N} \sum_{n=1}^{N} g_n^{(s)}$$

for N trajectories, ending in gain g_n; i.e., play the game N times, fill the value table,



then deploy the implicit policy.

Wait! But g_n *depends on* a policy (π)! We require as a parameter the very thing we're searching for?

Monte Carlo RL

Goal: Find $V^\pi(s) = \mathbb{E}[G_t \mid S_t = s]$ for all s. Then just change s for more value!

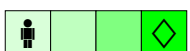
Problem (1): We don't have $V^\pi(s)$, we have to calculate (learn) it!

Problem (2): The expectation is annoying!

Solution: Monte Carlo Approximation!

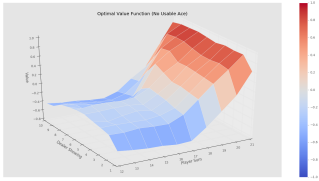
$$V^\pi(s) = \mathbb{E}[G_t \mid S_t = s] \approx \frac{1}{N} \sum_{n=1}^{N} g_n^{(s)}$$

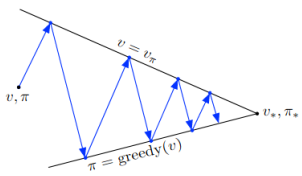
for N trajectories, ending in gain g_n; i.e., play the game N times, fill the value table,

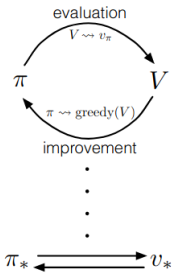


then deploy the implicit policy.

Wait! But g_n *depends on* a policy (π)! We require as a parameter the very thing we're searching for? **Iteration** and Exploration vs Exploitation ← key concepts!!!.



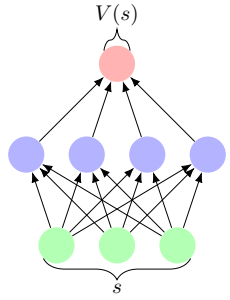




Source: Sutton and Barto, *Reinforcement Learning - An Introduction (2nd ed.)*, 2020

Scaling Up

Let $V_\theta(s)$ be a deep neural network parametrised by θ (also determines policy, π_θ).



Can be MLP, CNN, ...; s can be an image, signal, text, ...

At n-th training iteration:

$$\theta \leftarrow \theta + \alpha \left(g_n^{(s)} - V_\theta(s) \right) \nabla_\theta V_\theta(s)$$

This is just standard gradient descent; can be handled my any modern framework, e.g., PyTorch; where $g_n^{(s)}$ is our target ('class label').

Practical Challenges (And Lessons from Data-Stream Learning)

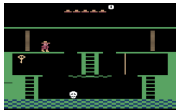
Typical instances of RL *are also* instances of data-stream learning (learning, i.e., updating θ, must be carried out incrementally).

From the data stream learning literature, many problems already discovered, e.g.,

- Cold-start problem (how to learn from $t = 0$); what assumptions can we make?
- What does t even mean? What if we miss some t?
- Instability issues; when/how to tune hyper-parameters?
- Online or batch-incremental learning? How fast do we need to learn?
- How to deal with delayed and sparse labels?
- Temporal dependence can ruins everything if not modeled/mitigated properly
- Robustness vs detecting and adapting to concept drift
- Catastrophic forgetting vs lack-of-capacity

See also: Read and Zliobaite, *Learning from Data Streams: An Overview and Update* 2023

Open Challenges in Reinforcement Learning



Even simple tasks (for a human) are currently unsolved in the true RL sense. And learning in real-world environments presents particular challenges.

- Learning quickly with limited samples; even when high-dimensional and complex
- Generalization and transfer: towards a foundation model
- Robustness and safety issues (offline RL and model-based RL)
- Explainable agents, interpretable actions
- Delayed/partial/weak observations, and handling the associated uncertainty
- Legal and ethical concerns, and alignment issues (agent's reward function vs ours)

In games and LLMs: Useful sometimes, failures acceptable, often amusing.
In the real world: agents must be trustworthy, acceptable results $\approx 100\%$ of the time.

Questions and Discussion

Questions

?

Resources and Further Reading

These slides:
`https://www.lix.polytechnique.fr/~jread/talks/2024_12_10-Malaga.pdf`

Suggested textbook: Sutton and Barto, *Reinforcement Learning - An Introduction (2nd ed.)* – particularly Chapter 3 and Chapter 4.

Testbed environments:

- `https://pypi.org/project/gymnasium/`
- `https://github.com/instadeepai/jumanji`
- `https://pettingzoo.farama.org/environments/classic/rps/` (multi-agent)

Baseline RL-algorithm implementations:

- `https://stable-baselines.readthedocs.io`

If you have more questions:
`jesse.read@polytechnique.edu`

`https://jmread.github.io/`

Introduction to Reinforcement Learning

Aprendizaje Automático del máster *Ingeniería del Software e Inteligencia Artificial*

Jesse Read

Version: December 10, 2024