Advanced Machine Learning and Autonomous Agents

Reinforcement Learning I

Jesse Read

Version: January 28, 2025

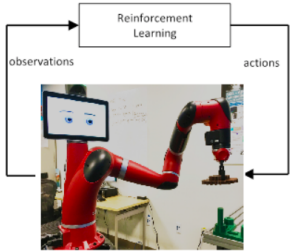
Goals for Today

Week 1: Deciding on actions given a knowledge representation
Week 2: Learning the knowledge representation, and complex inference
Week 3: Sequential Decision Making (Exploration-Exploitation tradeoff)
Today: Our decisions change the state of the environment (Reinforcement Learning)

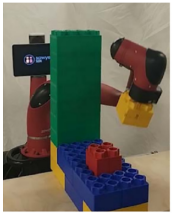
Objectives:

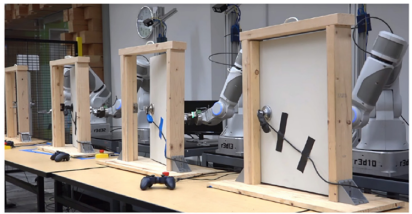
1. Understand the main concepts behind Reinforcement Learning
2. Get a feel for setting up RL problems in practice (warning: can be more subtle)

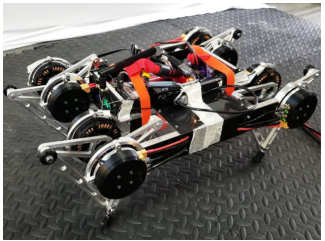
Applications: Robotics and Autonomous Vehicles



▸ Helicopter ▸ Robot ▸ More Robots



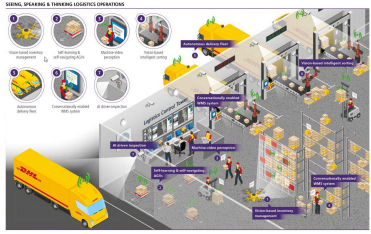




Ibarz et al., *How to train your robot with deep reinforcement learning: lessons we have learned*, 2021

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

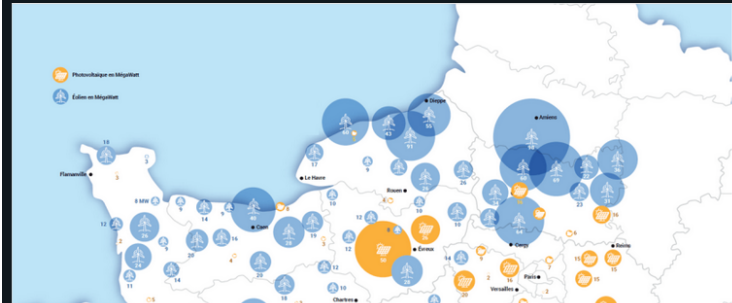


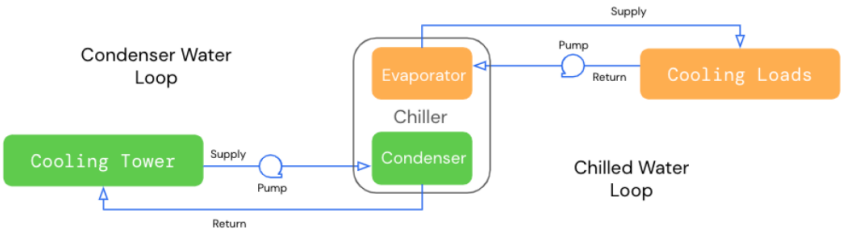
Application: Energy Systems

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

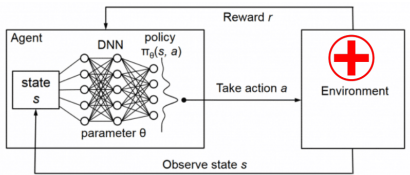
A small efficiency gain for every wind turbine would lead to a big boost globally

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

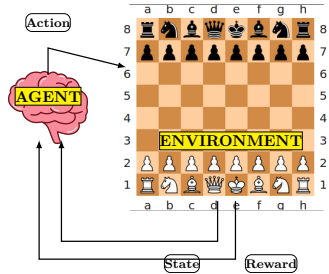


- Large Language Models (and other deep neural networks); HLRL etc.

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')

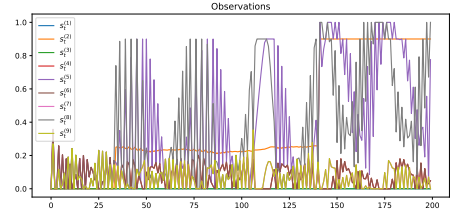


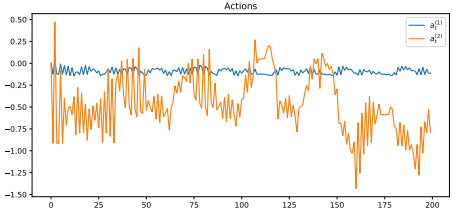
Important: The agent may alter the environment (and thus its own future state observations *and reward*!)

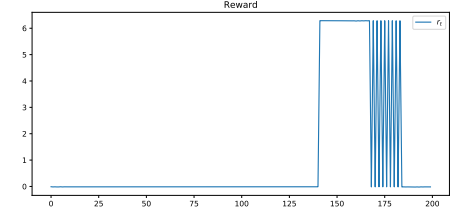
Environment-agent interaction (simplified):

```
env = Environment()
agent = Agent(env)
s = env.init()

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


Observations
$s_t^{(1)}$
$s_t^{(2)}$
$s_t^{(3)}$
$s_t^{(4)}$
$s_t^{(5)}$
$s_t^{(6)}$
$s_t^{(7)}$
$s_t^{(8)}$
$s_t^{(9)}$
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200


Actions
$a_t^{(1)}$
$a_t^{(2)}$
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
0
25
50
75
100
125
150
175
200


Reward
r_t
6
5
4
3
2
1
0
0
25
50
75
100
125
150
175
200

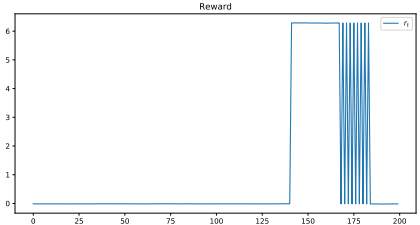
The Reward Signal and Trajectories

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?



The Environment

The environment provides interaction to the agent. Most importantly, it provides the environment dynamics, which determine how the agent can interact.

$$s' = f(s_t, a_t) \quad \triangleright \text{ Dynamic function}$$
$$r = r(s_t, a_t) \quad \triangleright \text{ Reward function}$$

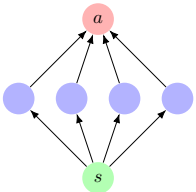
Note: often our intuition on real-world boundary between agent and its environment does not correspond to the modelisation of the problem.

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

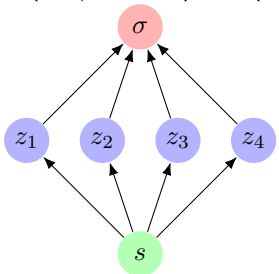
```
def act(s):
    if s < 3:
        # Go Right
        return 1
    else:
        # Go Left
        return -1
```

'Hand-coded' policy

A stochastic policy (provides a conditional probability distribution):

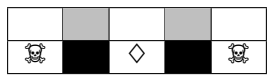
$$a_t \sim \pi(\cdot \mid s_t)$$

i.e., $P(A_t \mid S_t = s_t) = \pi(s_t)$.



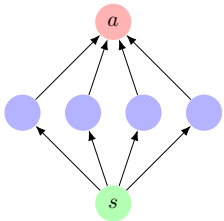
State s_t	$\pi(a_t = 1 \mid s_t)$ σ	$\pi(a_t = 0 \mid s_t)$ $1 - \sigma$
0	0.1	0.9
1	0.5	0.5
2	1	0
3	0.2	0.8

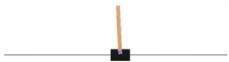
- Built-in exploration
- In competitive games, easier to defeat deterministic agents
- And aliased states: States that appear the same, but are not



Policy Search

Direct Policy Search (Black Box Optimization)





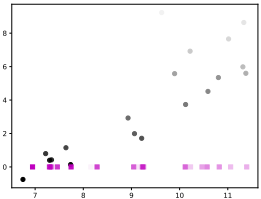
Suppose a parametric policy, parameters θ (e.g., of a neural network agent); i.e.,

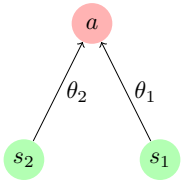
$$a_t = \pi_\theta(s_t)$$

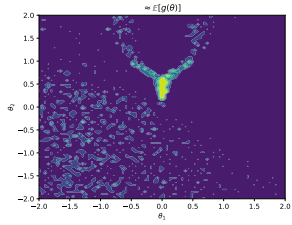
we can search directly in parameter space, i.e., policy search:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}}\, g(\theta)$$

where $g(\theta)$ indicates how good π_θ is, e.g., $g(\theta) = 10 \Leftrightarrow$ agent survived 10 seconds / obtained score of 10 at end of episode.







Move the purple paddle left ($a_t < 0$) or right ($a_t > 0$) to catch a falling black ball (coordinates $\boldsymbol{s}_t = [s_1, s_2]$)

- Many off-the-shelf search & optimization tools are available
- Works 'embarrassingly well' on some problems
- Handles continuous state/action spaces naturally
- No need to study reinforcement learning

But

- Expensive to evaluate $g(\theta)$ (run simulations/play games)
- Noisy (large variance); randomness from the environment!
- Curse of dimensionality: optimization in high dimensions (> 100) is very hard – it's why we don't train classifiers with black-box optimization!

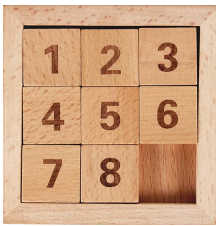
Fully Observed Deterministic Environments

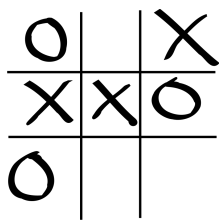
Fully Observed Deterministic Environments

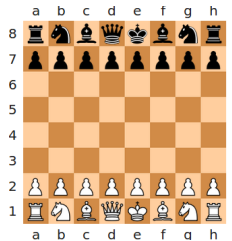
In deterministic environments: for any given action, the next state is known for certain.

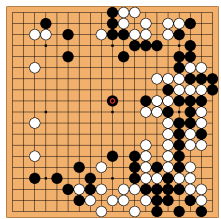
In fully observed environments: complete view of the environment at time t.

Examples: 8-Puzzle, Noughts & Crosses (Tic Tac Toe), Chess, ...









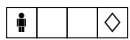
Environment-Agent Interaction in Deterministic Environments

$a_0 \leftarrow \pi(s_0)$ ▷ Initial action
for $t = 1, 2, \ldots$ **do**
  $s_t \leftarrow f(s_{t-1}, a_{t-1})$ ▷ Act, observe
  $r_t \leftarrow r(s_{t-1}, a_{t-1})$
  $a_t \leftarrow \pi(s_t)$ ▷ Decide action
end for

- fully-observed case: full knowledge of f, and full view of s_t!
- reward function r also known

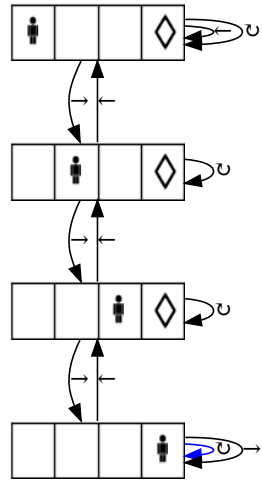
Toy Example 1

- state $s \in \{1, 2, 3, 4\}$
- action $a \in \{\leftarrow, \circlearrowleft, \rightarrow\}$
- reward function $r(4, \circlearrowleft) = 1$
  else $r(s, a) = 0$ for other values of s, a
- dynamics fn $s' = f(s, a)$ depicted by *transition graph* (right)

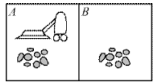
e.g., $s_t =$ 

 $= f($  $, \rightarrow)$

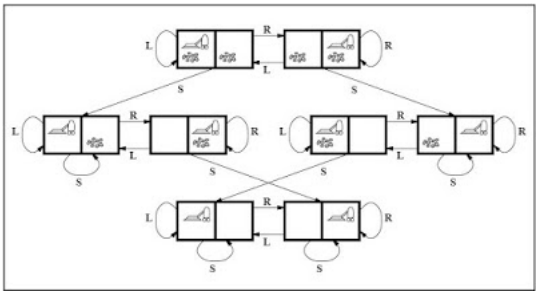
And `end of episode` at t when $\circlearrowleft = \pi(4)$



Vacuum Cleaner World:



There are 8 possible states:



From Russel and Norvig, *Artificial Intelligence: A Modern Approach (2nd ed.)* 2002

The 8-Puzzle:

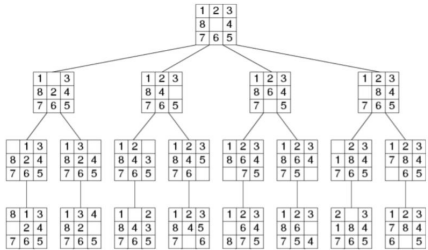
1	3	5
7	2	4
8		6

$\Rightarrow$

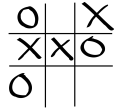
1	2	3
4	5	6
7	8	

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

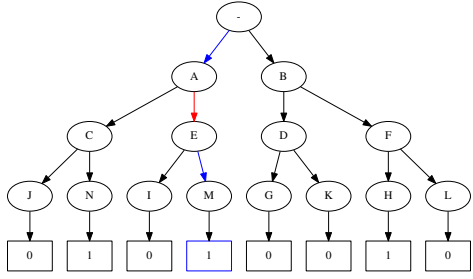
1. Generate the search tree to goal state
2. Apply payoff to leaf
3. Backup
4. Choose the best branch



Adversaries



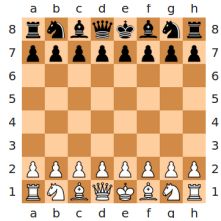
What about the other player (adversary)? Can be made deterministic via minimax algorithm,



where the adversary is the ‘min’. Assumptions:

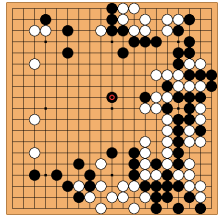
Scaling Up

What if the branching factor is too large and/or long path to goal?



Use advanced/approximate-search techniques.

What if branching factor is really large?



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:

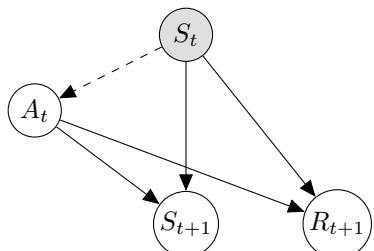
Image credits: the internet

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 dynamics
- $\pi(a \mid s)$ policy of the agent



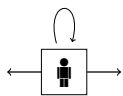
In Reinforcement Learning we do not necessarily know this model.

Slippery Floor Environment

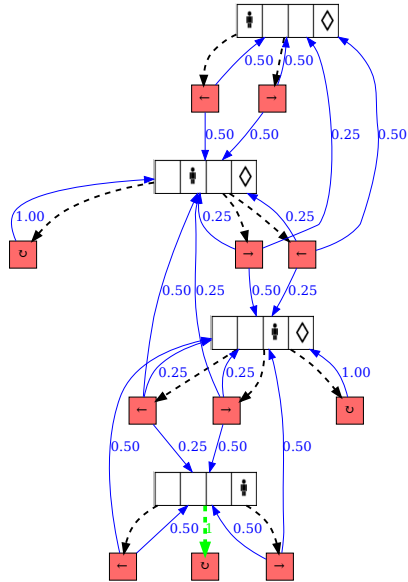
- state $s \in \{1, 2, 3, 4\}$
- action $a \in \{\leftarrow, \circlearrowleft, \rightarrow\}$
- reward function $r(4, \circlearrowleft) = 1$
  else $r(s, a) = 0$ for other values of s, a
- transition dynamics $s' \sim p(\cdot \mid s, a)$

e.g., $p($  $\mid$  $, \leftarrow) = 0.5$

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



s



(*transition graph* representing MDP dynamics)

The Markov Property

MDPs provide a framework for modeling sequential decision making in stochastic environments. Most reinforcement learning problems can be framed as an MDP.

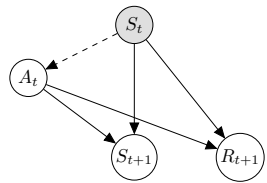
M The Markov property

D We add decisions (motivated by rewards)

P A [Markov] process

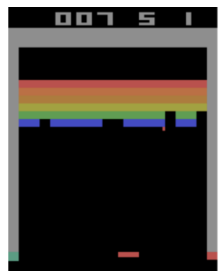
Markov property: The effects of action a_t from state s_t depend only on that action and state; i.e.,

$$p(s_{t+1} \mid s_t, a_t) = p(s_{t+1} \mid s_1, \ldots, s_t, a_1, \ldots, a_t)$$



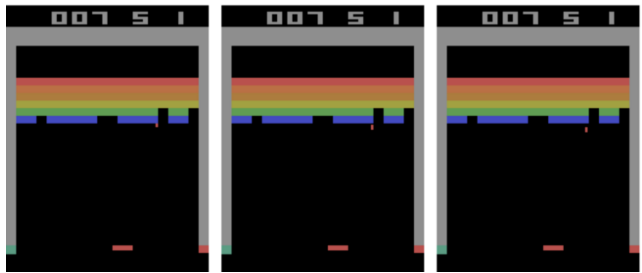
Implications: if Markov, it's possible for the agent to act optimally knowing (observing) state s_t only.

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**!) ??? ...

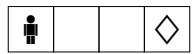
to optimize what?

The Gain (Finite Scenario)

The gain (aka return[1] 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.



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

The Gain (Finite Scenario)

The gain (aka return[1] 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 $r_{t+1000} = 1$?

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

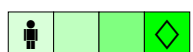
The Return/Gain (Infinite Scenario)

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!

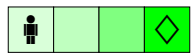
The Return/Gain (Infinite Scenario)

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!

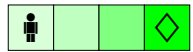
The Return/Gain (Infinite Scenario)

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

Motivation and Role of the Discount Factor γ

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-1} R_T$$

- Reward today is better than reward tomorrow
- The future is increasingly uncertain (difficult to represent)
- The agent may die at any time
- $\gamma \in (0, 1)$ determines the present value of future rewards
- The reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received now
- If γ is close to 0, the agent is 'myopic': maximize immediate rewards
- If γ approaches 1, the agent is more 'far-sighted', takes future rewards into account more strongly (even at the cost of missing out on short-term rewards)
- In practice: γ can have a significant impact on agent performance

The Value Function

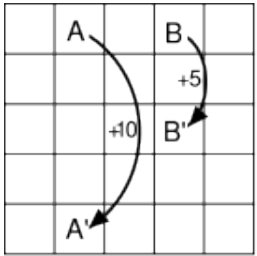
The value function (aka state-value function),

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

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 $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$:

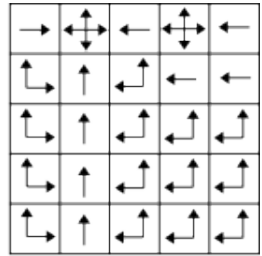
	s_1	s_2	s_3	s_4
V^{π}				



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

Ref: Sutton and Barto. *Reinforcement Learning: An Introduction*, 2018.

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

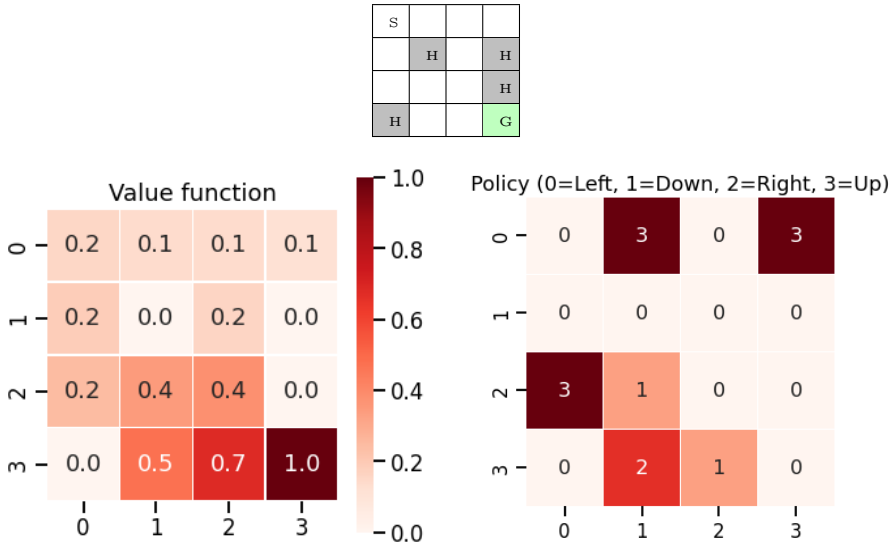


Image credits: Jérémie Decock

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^π	a_1	a_2
s_1		
s_2		
s_3		
s_4		

The advantage of Q over V is that it can tell us what actions are available from s.
The greedy policy is explicit:

$$a_t = \pi(s_t) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s_t, a)$$

Bellman Equation for V^π (Preliminaries)

Recall (from Slide 41) the value function:

$$V^\pi(s) = \mathbb{E}[G_t \mid S_t = s] \tag{3}$$

A recursion trick we will need:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \quad \triangleright \text{ Recursion!} \end{aligned} \tag{4}$$

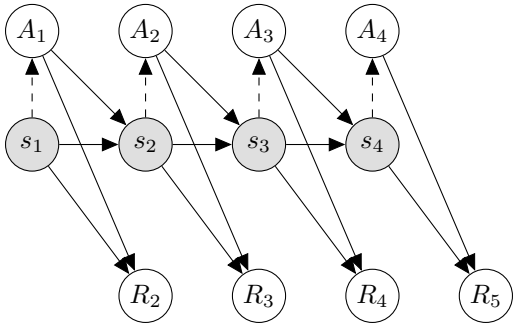
A reminder on conditional probability and expectations and marginalizing out:

$$\mathbb{E}_{A,R,S'}[g(A,R,S') \mid S = s] = \sum_{a,r,s'} g(a,r,s')p(a,r,s' \mid s)$$

The expectation is with respect to A, S.

Total law of expectation:

$$\mathbb{E}[G] = \mathbb{E}_S[\mathbb{E}[G|S]]$$


A_1
A_2
A_3
A_4
S_1
S_2
S_3
S_4
R_2
R_3
R_4
R_5

Bellman Equation V^π (Derivation)

From an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$: $\forall s \in \mathcal{S}$, under policy π:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] && \triangleright \text{ Cond. expectation – cf. Eq. (3)} \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \triangleright \text{ Recursion; from Eq. (4)} \\
&= \mathbb{E}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \mathbb{E}_{a_t \sim \pi(\cdot \mid s_t)}\left[r(s_t, a_t)\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \sum_{a,s'} \underbrace{p(a \mid s) p(s' \mid s, a)}_{p(a,s' \mid s)} \left[r(s,a) + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] && \triangleright \text{ Marginalise out} \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \Big[r(s,a) + \gamma \underbrace{\mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]}_{V^\pi(s')} \Big]
\end{aligned}
\tag{5}
\tag{6}
$$

Where Eq. (6) completes the recursion via Eq. (5).

Why is this important?

Bellman Equation V^π (Derivation)

From an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$: $\forall s \in \mathcal{S}$, under policy π:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] && \triangleright \text{ Cond. expectation – cf. Eq. (3)} \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \triangleright \text{ Recursion; from Eq. (4)} \\
&= \mathbb{E}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \mathbb{E}_{a_t \sim \pi(\cdot \mid s_t)}\left[r(s_t, a_t)\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \sum_{a,s'} \underbrace{p(a \mid s) p(s' \mid s, a)}_{p(a,s' \mid s)} \left[r(s,a) + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] && \triangleright \text{ Marginalise out} \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \Big[r(s,a) + \gamma \underbrace{\mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]}_{V^\pi(s')} \Big] && (6)
\end{aligned}
\qquad (5)
$$

Where Eq. (6) completes the recursion via Eq. (5).

Why is this important?

A [closed form] solution for expected return/value!

Bellman Equation V^π (Derivation)

From an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$: $\forall s \in \mathcal{S}$, under policy π:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[G_t \mid S_t = s\right] && \triangleright \text{ Cond. expectation – cf. Eq. (3)} \\
&= \mathbb{E}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] && \triangleright \text{ Recursion; from Eq. (4)} \\
&= \mathbb{E}\left[R_{t+1} \mid S_t = s\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \mathbb{E}_{a_t \sim \pi(\cdot \mid s_t)}\left[r(s_t, a_t)\right] + \gamma \mathbb{E}[G_{t+1} \mid S_{t+1} = s'] \\
&= \sum_{a,s'} \underbrace{p(a \mid s) p(s' \mid s, a)}_{p(a,s' \mid s)} \left[r(s,a) + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\right] && \triangleright \text{ Marginalise out} \\
&= \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} p(s' \mid s, a) \Big[r(s,a) + \gamma \underbrace{\mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]}_{V^\pi(s')}\Big] && (6)
\end{aligned}
\qquad (5)
$$

Where Eq. (6) completes the recursion via Eq. (5).

Why is this important?

A [closed form] solution for expected return/value!

However: what we really want (goal of reinforcement learning) is π_*.

$V \approx V^{\pi}$

Recall:

	s_1	s_2	s_3	s_4
V^{π}				

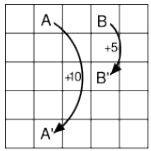
But what are the values? Where do we get them from?
We need to run policy π to find out!

But what is policy π? Where do we get it from?

We need to calculate V^{π} to find out!

Notation: If V^{π} provides π, then what we really want is v^*, it provides the *optimal* policy π^*.

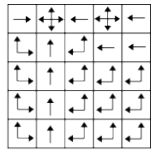
	s_1	s_2	s_3	s_4
v^*				



a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*


c) π^*

Bellman Optimality Equation v^* and π^*

So: we have an expression for $V^\pi(s)$.

The optimal policy (and the value function associated with it):

$$\pi^*(s) = \text{argmax}_\pi V^\pi(s) \quad \forall s \in \mathcal{S}$$
$$v^*(s) = \max_\pi V^\pi(s) \quad \forall s \in \mathcal{S}$$

The value of state s, following policy π (note the greedy policy) is

$$V^\pi(s) = Q^\pi(s, \pi(s)) = \max_{a \in \mathcal{A}(s)} Q^\pi(s, a)$$

It means that (from Eq. (6), $V^\pi(s)$):

$$V^\pi(s) = Q^\pi(s, \pi(s)) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma V^\pi(s') \right]$$

The action-value function Q shares the same optimal policy π^*.

$$
\begin{aligned}
v^*(s) &= \max_{a \in \mathcal{A}(s)} q^{\pi_*}(s, a) \\
&= \dots \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma v^*(s') \right] \\
\pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}(s)} q^{\pi_*}(s, a)
\end{aligned}
$$

Note, again: the max and the argmax.

$$
\begin{aligned}
v^*(s) &= \max_{a \in \mathcal{A}(s)} q^{\pi_*}(s, a) \\
&= \dots \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma v^*(s') \right] \\
\pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}(s)} q^{\pi_*}(s, a)
\end{aligned}
$$

Note, again: the max and the argmax.

Recall our main objective: how to get π^*? (policy which takes optimal actions)

Generalized Policy Iteration

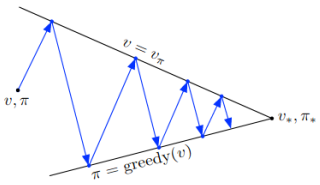
The bellman equation is typically solved iteratively.

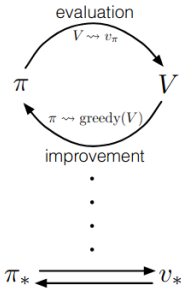
Goals:

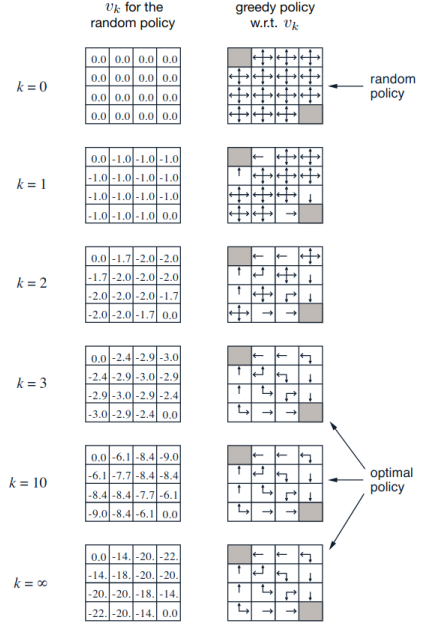
1. make the value function consistent with the current policy, and
2. make the policy greedy with respect to the current value function

Iterative solutions:

- Value iteration
- Policy iteration
- Q-learning
- Sarsa
- ...







Ref: Sutton and Barto. *Reinforcement Learning: An Introduction*, 2020.

Exploration vs Exploitation

From any state s, at time t, we want:

$$\underset{\pi}{\text{argmax}}\, \mathbb{E}_{s' \sim p(s), a \sim \pi(s)}[G_t]$$

In reinforcement learning, we do not know the underlying system dynamics (p) of the MDP, we need to learn it through interaction with the environment.

Easy: run π, gather data (s, a, r', s'), calculate/approximate $\mathbb{E}$.

Hard: how to run π, i.e., which π?

Exploration $\neq$ Exploitation

A Monte Carlo View

1. Play many episodes with a policy

$$a_t \sim \pi(\cdot|s_t)$$

   and record the gain from each (s_t, a_t)-pair
2. Use these samples to approximate the expectation:

$$Q^\pi(s,a) = \frac{1}{n}\sum_{i=1}^{n} g_{a|s}^{(i)} \approx \mathbb{E}[G \mid s, a]$$

3. Repeat many times. Then employ the greedy policy π on Q:

$$a_t = \underset{a\in\mathcal{A}(s_t)}{\operatorname{argmax}}\, \pi(a|s_t) = \underset{a\in\mathcal{A}(s_t)}{\operatorname{argmax}}\, Q^\pi(s_t, a)$$

Trade-off: Exploration vs exploitation.

Types of Reinforcement Learning Agents

Value-Based or Policy-Based ?

- Value Based (value function; policy is implicit)
- Policy Based (no value function)
- Actor Critic (both value function + policy)

Model Free vs Model Based: We can build a model of the environment (or not).

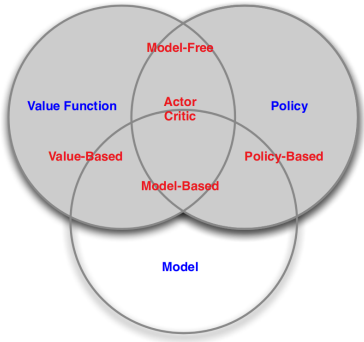


Image from [1]

Summary / Important Concepts

- Agent and Environment
- States, Actions, Reward
- Model of the environment (MDP)
  - Markov property
  - Deterministic vs stochastic
  - Unknown vs fully observable vs partially observable
- Return/Gain (finite horizon vs infinite horizon; γ)
- Policy (deterministic policy, stochastic policy, greedy policy)
- State Value and Action-Value functions
- Bellman Equations
- Exploration vs Exploitation trade-off

Reinforcement learning is difficult (compared to supervised learning), but there are many applications where supervised learning does not apply.

Deep Reinforcement Learning

[2]

Action space is simple (move left, right, shoot, ...), but state space is huge! Q-table impossible!

A Deep Representation of the Q-table

We replace the Q table with a neural network:

$$Q(s, a) = Q_{\boldsymbol{w}}(s)_a$$

where $Q_{\boldsymbol{w}}$ is a (deep) neural network parametrized by $\boldsymbol{w}$.

For example,

$$[Q(s, a_1), \ldots, Q(s, a_4)] = Q_{\boldsymbol{w}}(s)$$

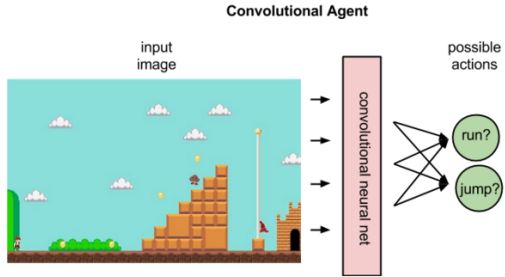
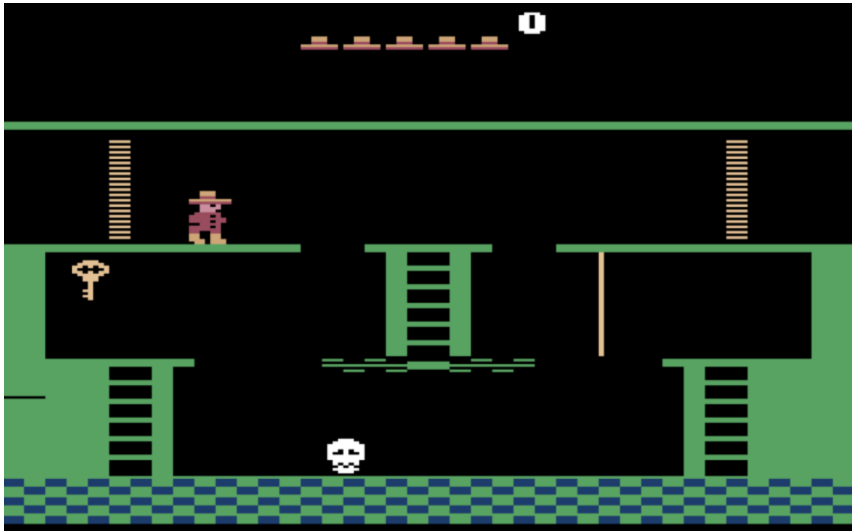


Image Source: [3]

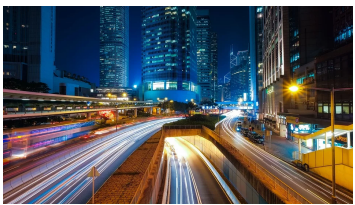
Montezuma's Revenge ('unsolved' by Reinforcement Learning)



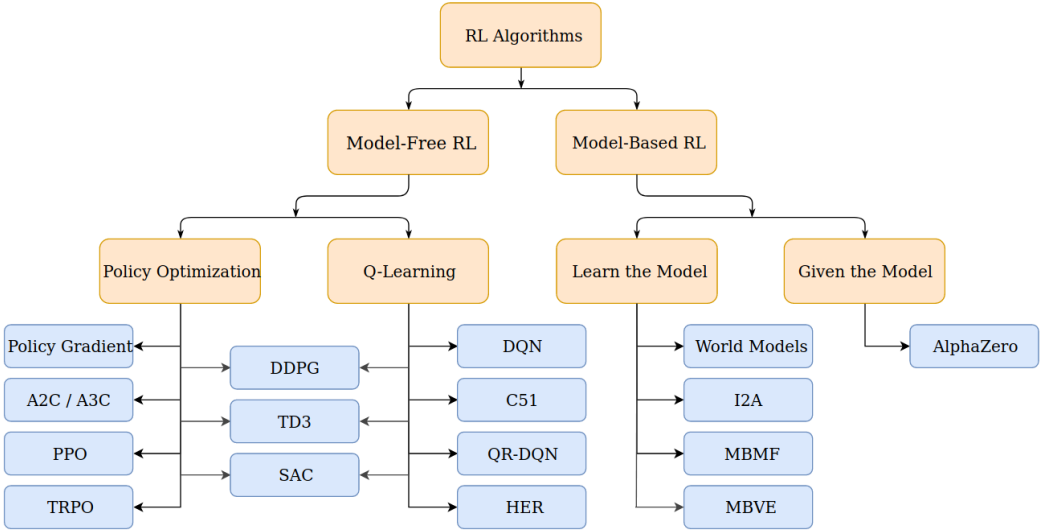
Montezuma's Revenge – Notoriously Difficult for RL Agents [4]

Modern 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
- Multi-objective reward fns, multi-agent, alignment
- Legal and ethical concerns, and alignment issues (agent's reward function vs ours)



A Taxonomy of Methods



Modern Machine Learning Methods [5]

Advanced Machine Learning and Autonomous Agents

Reinforcement Learning I

Jesse Read

Version: