Reinforcement learning: an introduction and some applications to spiking neurons

Frédérick Garcia

INRA Applied Mathematics and Informatics, Toulouse

November 29, 2019
Université Cote d'Azur, NeuroMod, Nice

Plan

Plan

Minicours objectives and contents

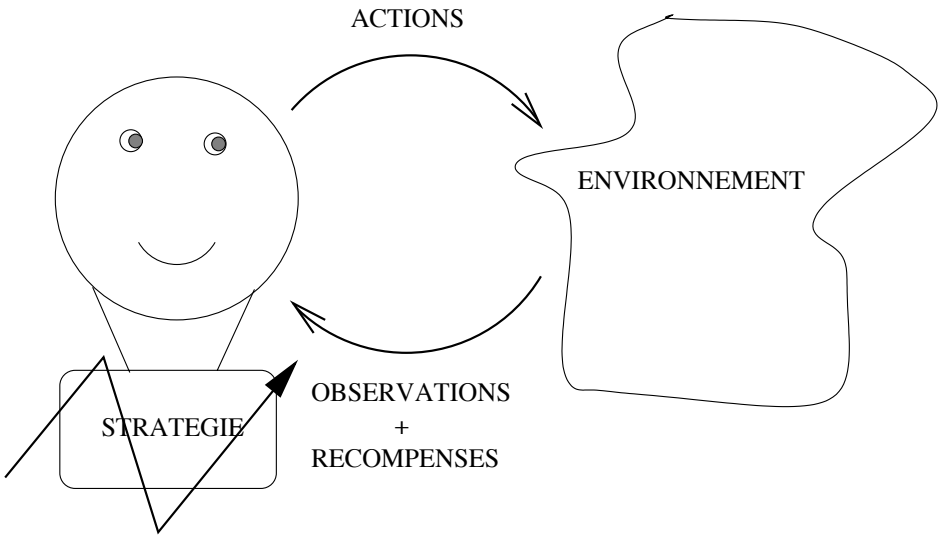
This course aims at introducing the Reinforcement Learning approach and its application to learning in spiking networks.

Course 1
- Introduction
- Sequential decision making under uncertainty
- Markov Decision Problems in finite and infinite horizons
- Reinforcement Learning

Course 2
- Generalization in Reinforcement Learning
- Reinforcement Learning and Spiking Neural Networks

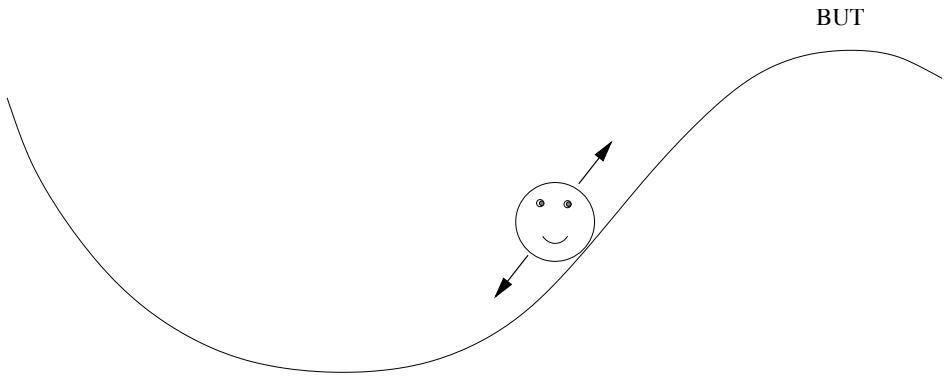
The problem of reinforcement learning

An autonomous agent exploring his environment, in order to learn a behavior that maximizes the rewards he receives.



What types of applications ?

RL started with control problems, quite toy problems, like the mountain car :

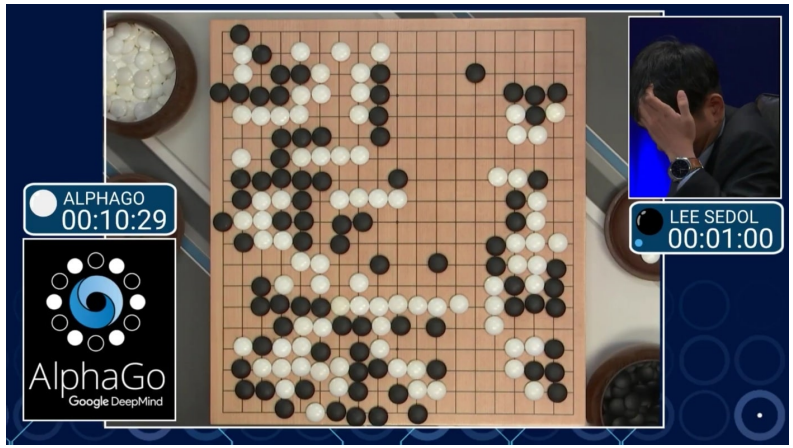


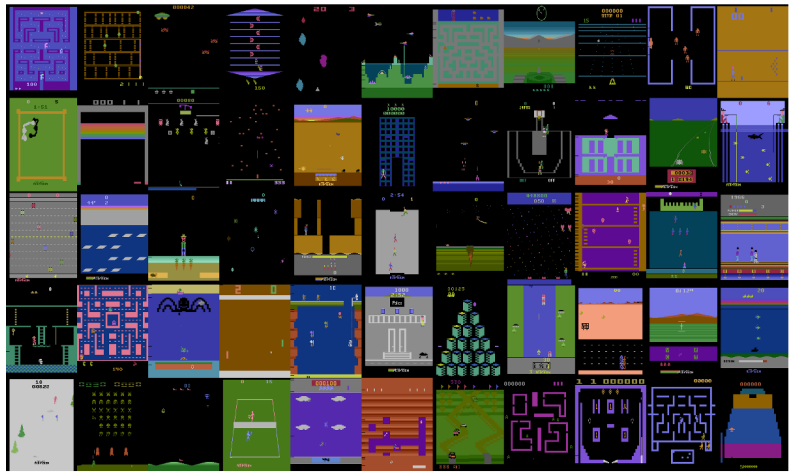
- the car can accelerate in both directions
- the goal is to reach the top of the hill, as fast as possible

What types of applications ?

Historically, and more recently, games :

- checkers, backgammon
- tetris
- go, chess
- arcade games (atari2600)

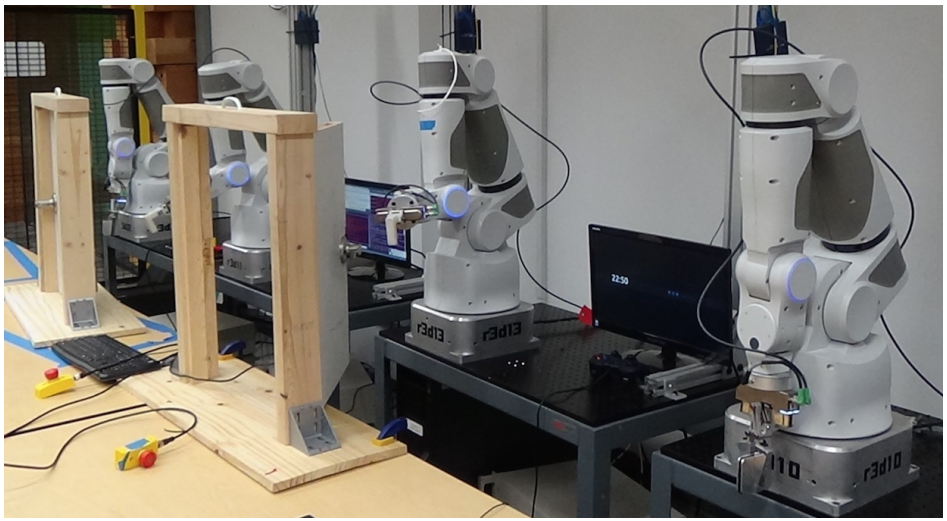




What types of applications ?

Also very typical, autonomous robotics :

- robotic arms, factory robotics
- mobile robotics, autonomous car driving
- humanoid robots



What types of applications ?

Recently, more serious applications :

- allocation and scheduling of ressources
- traffic light control
- web system configuration
- bidding and advertising
- autonomous car driving
- crop management

Twofold objective of RL

- Control the systeme optimally
- Learn this optimal policy by trial and error

First point well studied with the theory of optimal control and sequential decisions under uncertainty (Hamilton-Jacobi-Bellman or Bellman equations, 1954)

Reinforcement learning (Samuel 1959, Minsky 1964, Widrow 1973, Holland 1975, Sutton 1984) is an extension of these works to the online paradigm, for very large problems

Plan

decision theory

Decision theory is about human decision making in a world of incomplete information.
In decision theory, a cognitive decision maker plays against a randomizing nature.

Expected Utility (EU) theory, introduced by von Neuman and Morgenstern in 1944, is the dominant theory of economic choice

decisions theory

The three fundamental concepts of decision theory are acts, states of nature and outcomes.

- Acts are under the control of the decision maker, who has to choose one act from a given set of possible acts $A = \{a_1, a_2, ...a_A\}$

decisions theory

The three fundamental concepts of decision theory are acts, states of nature and outcomes.

- Acts are under the control of the decision maker, who has to choose one act from a given set of possible acts $A = \{a_1, a_2, ...a_A\}$
- States of nature are under the control of nature, and are probabilistically selected by nature from a set $S = \{s_1, s_2, ...s_S\}$. They represent the circumstances about which there is uncertainty

decisions theory

The three fundamental concepts of decision theory are acts, states of nature and outcomes.

- Acts are under the control of the decision maker, who has to choose one act from a given set of possible acts $A = \{a_1, a_2, ...a_A\}$
- States of nature are under the control of nature, and are probabilistically selected by nature from a set $S = \{s_1, s_2, ...s_S\}$. They represent the circumstances about which there is uncertainty
- When the act has been chosen and the state has been selected, the outcome $o = F(a, s)$ is determined, where F is the outcome mapping. The set of possible outcomes is $O = F(A, S) = \{F(a, s) \mid a \in A,\ s \in S\}$

Acts, states and outcomes can be complex objects, but decision theory only consider them as set elements.

state and outcome probabilities

In decision making under risk, or decisions under uncertainty, we assume that for each possible act a, the decision maker is able to define a probability distribution P_a over the states of nature :

$$P_a = \{p_1, p_2, ..., p_S\}, \text{ with } p_i = P(s_i \mid a) \text{ and } p_1 + p_2 + ... + p_S = 1$$

This leads to probability distributions Q_a over the the set of possible outcomes :

$$Q_a = \{q_1, q_2, ..., q_O\}, \text{ with } q_j = P(o_j \mid a) = \sum_i p_i \mid o_j = F(a, s_i)$$

Outcomes, and Lotteries

Decision theory assumes that outcomes are the only thing decision maker cares about when taking its decision. So it is assumed that the decision maker has preferences over outcomes and only over outcomes.

In decision theory, an action is thus represented by a lottery :

$$L = \{O, Q\},$$

with $O = \{o_1, o_2, ..., o_O\}$, $Q = \{q_1, q_2, ..., q_O\}$ and $\sum_j q_j = 1$

Axioms of utilty theory constrain the possible patterns of preference between lotteries a decision maker can exhibit.

The expected utility theorem

Any preference pattern following these axioms can be represented by a real-valued function U on the set of outcomes O, such that

$$L_1 \succeq L_2 \text{ iff } U(L_1) \geq U(L_2)$$

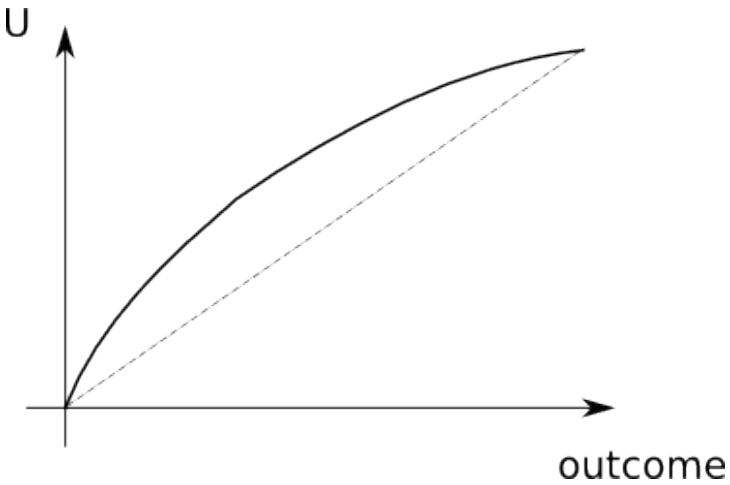
with

$$U(\{\{o_1, o_2, ..., o_O\}, \{p_1, p_2, ..., p_O\}\}) = \sum_i p_i U(o_i)$$

The utility value function and attitudes toward risk

The utility function U is unique up to a positive linear transformation $U' = aU + b$
For instance, $U(o_b) = 1$ and $U(o_w) = 0$

For continuous outcomes (money, time, etc.) U can be normally described by a continuous function $u(x)$



U concave : risk averse
$u(\sum_i p_i o_i) \geq \sum_i p_i u(o_i)$

U convex : risk seeking
$u(\sum_i p_i o_i) \leq \sum_i p_i u(o_i)$

U linear : risk neutral
$u(\sum_i p_i o_i) = \sum_i p_i u(o_i)$

Maximising expected utility

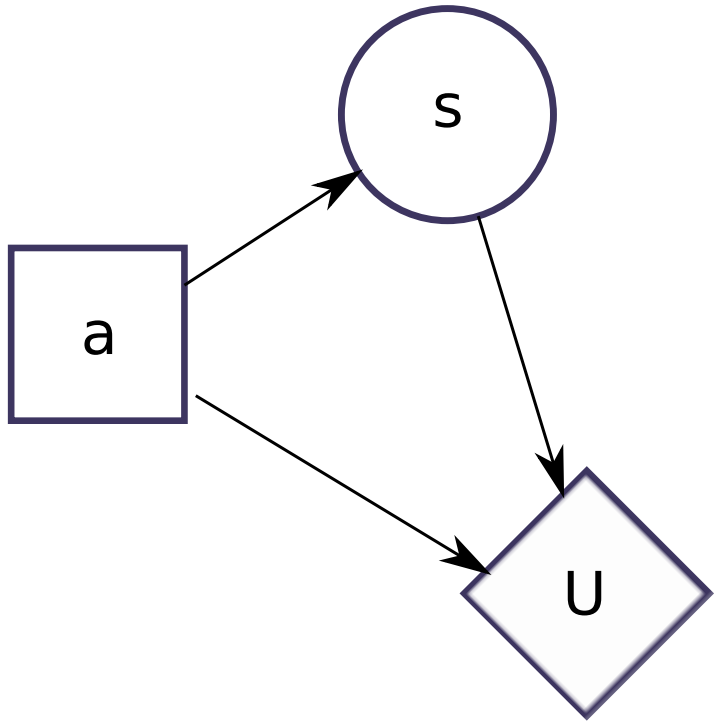
With the EU theory, the best action is the one that maximizes the utility of its corresponding lottery

$$
\begin{aligned}
U_{a^*} &= \max_{a \in A} U(L_a) \\
&= \max_{a \in A} U(\{O, Q_a\}) \\
&= \max_{a \in A} \sum_j P(o_j \mid a) U(o_i) \\
&= \max_{a \in A} \sum_i P(s_i \mid a) U(F(a, s_i)) \\
&= \max_{a \in A} E[U(F(a, s) \mid a)]
\end{aligned}
$$

One-step decision making

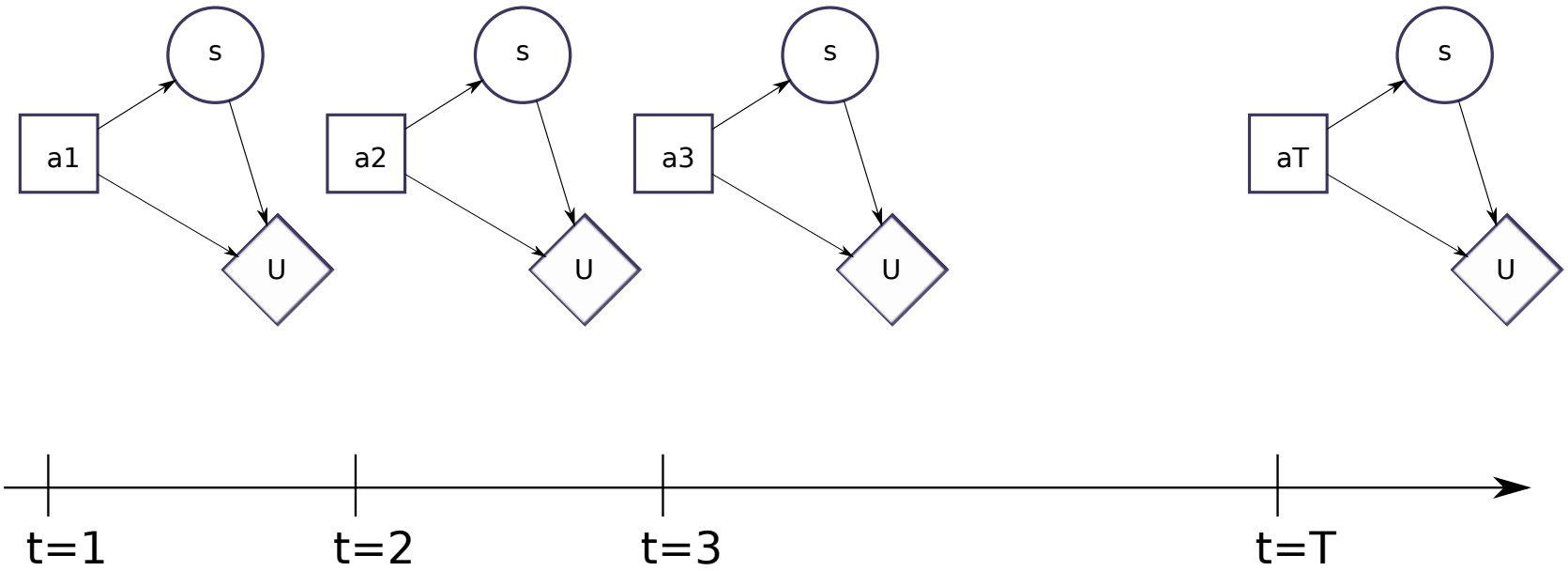
Decision theory is about one-step decision making

$$U_{a^*} = \max_{a \in A} E[U(F(a, s) \mid a)]$$



Multi-step decision making

The theory of sequential decision making under uncertainty is about multi-step decision making

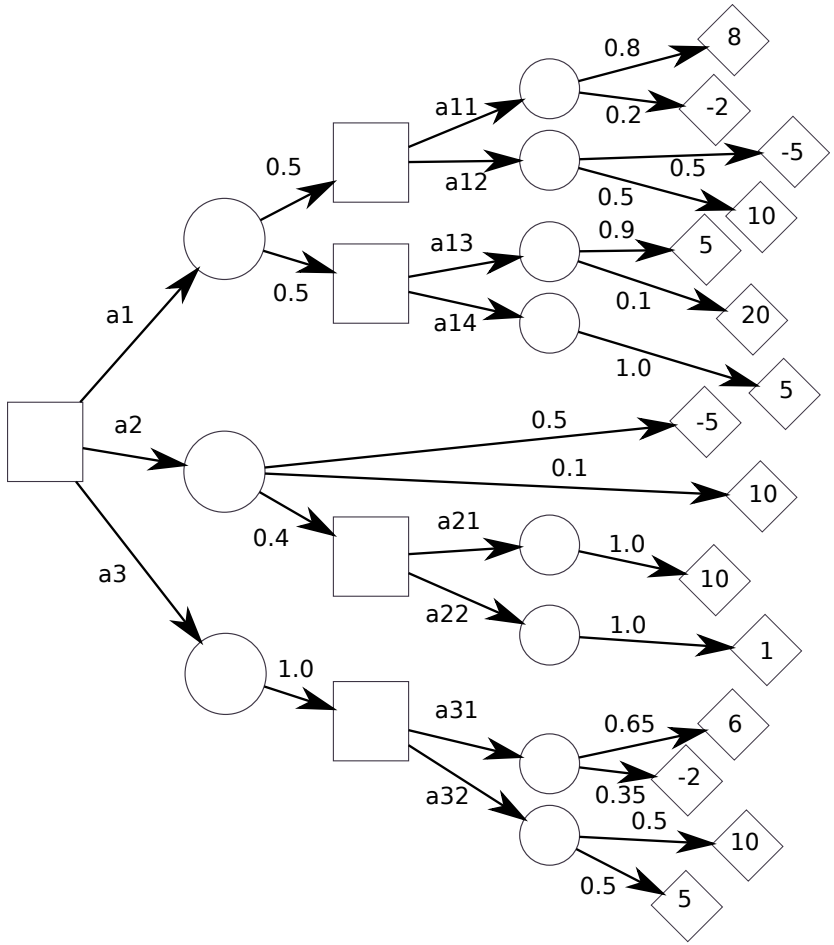


Some new problems to consider

- A sequential decision making problem is not just a set of independent one-step decision making problems. How to formalize the relations between these individual problems ?
- How to aggregate the utilities of successive outcomes ? What new choice criteria can we consider ?
- How to represent decisions in sequential setting, what kind of optimal solutions can we expect : sets of acts, plans, decision rules, policies, behaviors... ?

Planning with decision trees

An explicit tree-based representation of all possible scenarios (tree paths) from the initial situation (root) to final outcomes (leaves), through decision nodes and chances nodes



Solving decision trees

A foldback analysis, from the leaves to the root

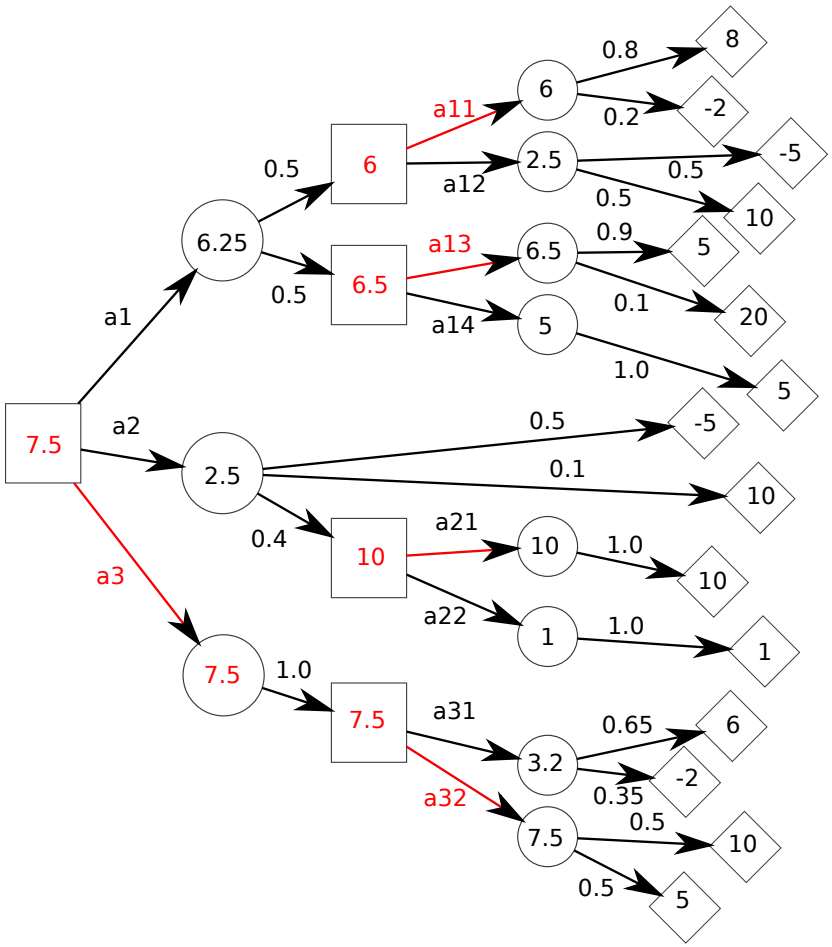
- the utility value of a leaf is its outcome's value $U(l)$
- the utility value of a chance node is the expected utility of its successors

$$U(c) = \sum_{n \in succ(c)} p(c \rightarrow n) U(n)$$

- the utility value of a decision node is the maximum utility of its successors

$$U(d) = \max_{n \in succ(d)} U(n)$$

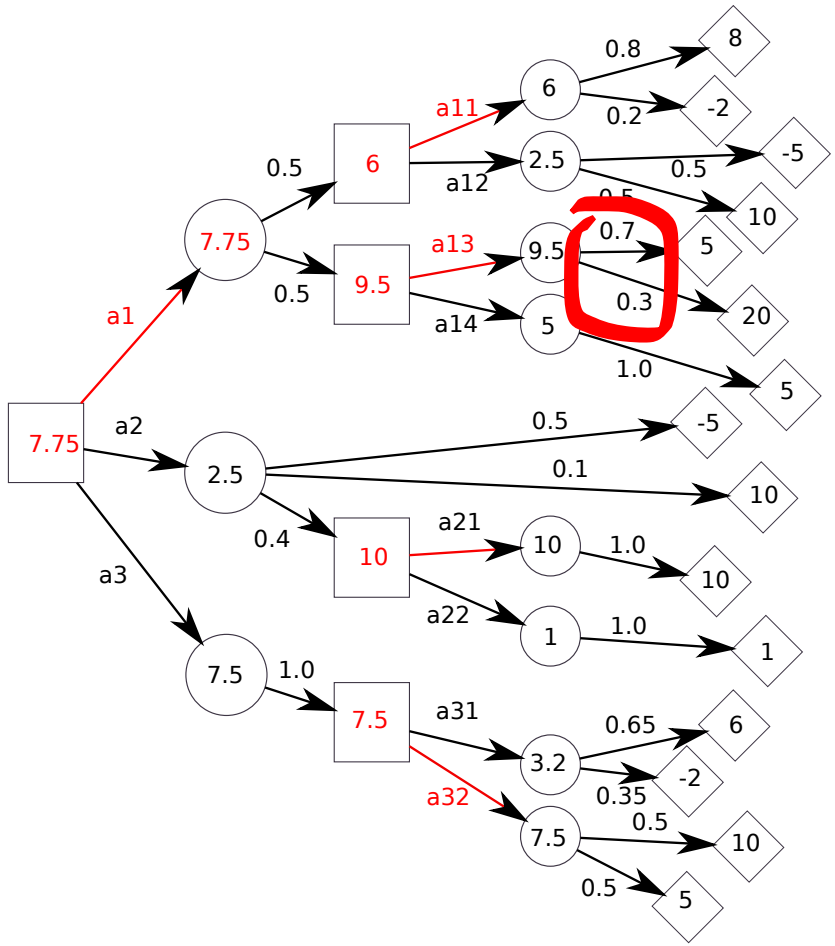
Solving decision trees



Here the optimal solution is the sequence $< a_3, a_{32} >$, with $U^* = 7.5$

Solving decision trees

Changing some probabilities or outcome values can lead to different optimal solution structures...



The new optimal solution is a conditional plan : $< a_1, \begin{array}{l} node_{11} \rightarrow a_{11} \\ node_{12} \rightarrow a_{13} \end{array} >$

Limits of decision trees

- do not allow independence or correlation relations to be exploited
- grow exponentially with problem size, complexity in $O((AO)^T)$
- unnatural representation of conditional probabilities

Markov Decision Processes are a better framework !

Plan

What are MDPs ?

MDP = Markov Decision Processes, or also Markov Decision Problems (Putterman 1994)

1. A mathematical framework for modeling optimal decision problems
2. A family of algorithmic methods for solving these optimization problems

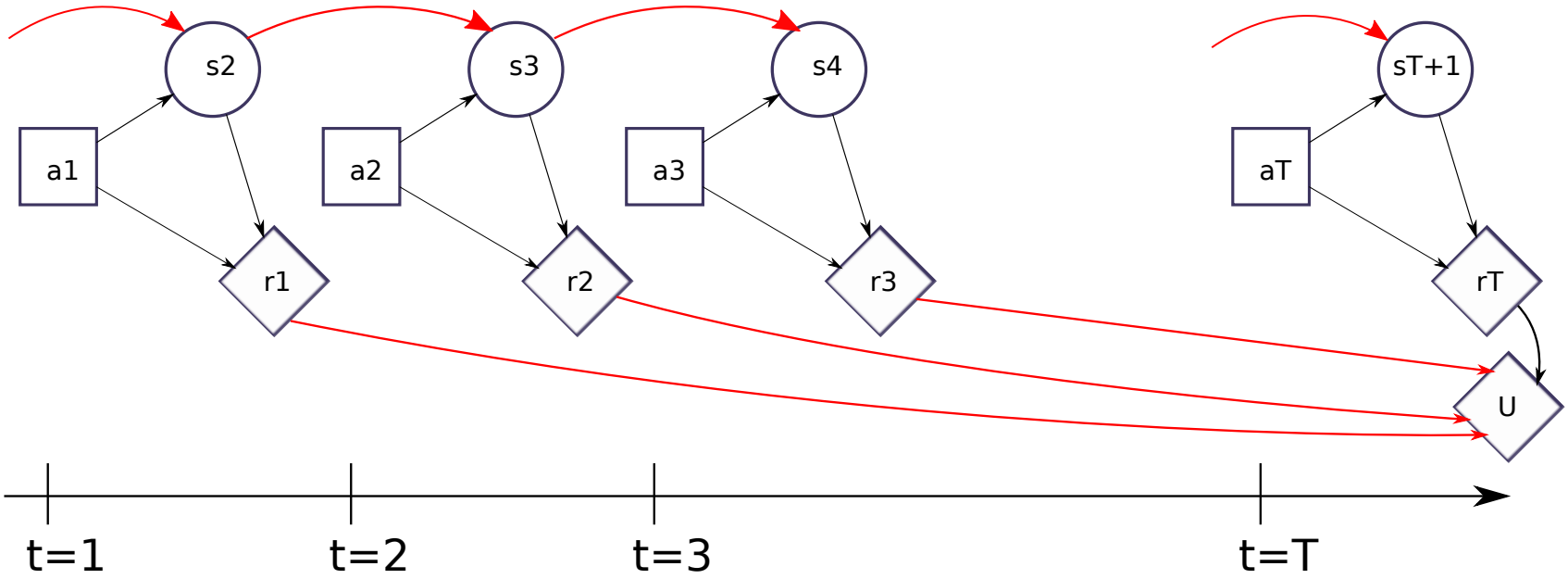
A simple and efficient modelling and optimisation framework, a theoretically grounded decision approach, an extendable formalism that can be modified and adapted.

From decision trees to MDP

With finite-horizon MDP, decision trees are transformed so that

- to each decision node is associated a state, the state of the system just before taking the decision
- chance nodes and random events ω are represented by transitions between states
- state transition probabilites are assumed to be Markovian
- rewards are associated to state transitions, not only to final states (leaves), and rewards are additive
- optimal solutions are decision rules $state_i \rightarrow action_j$ at each decision stage $t = 1, ..., T$

Coupling one-step decision making problems

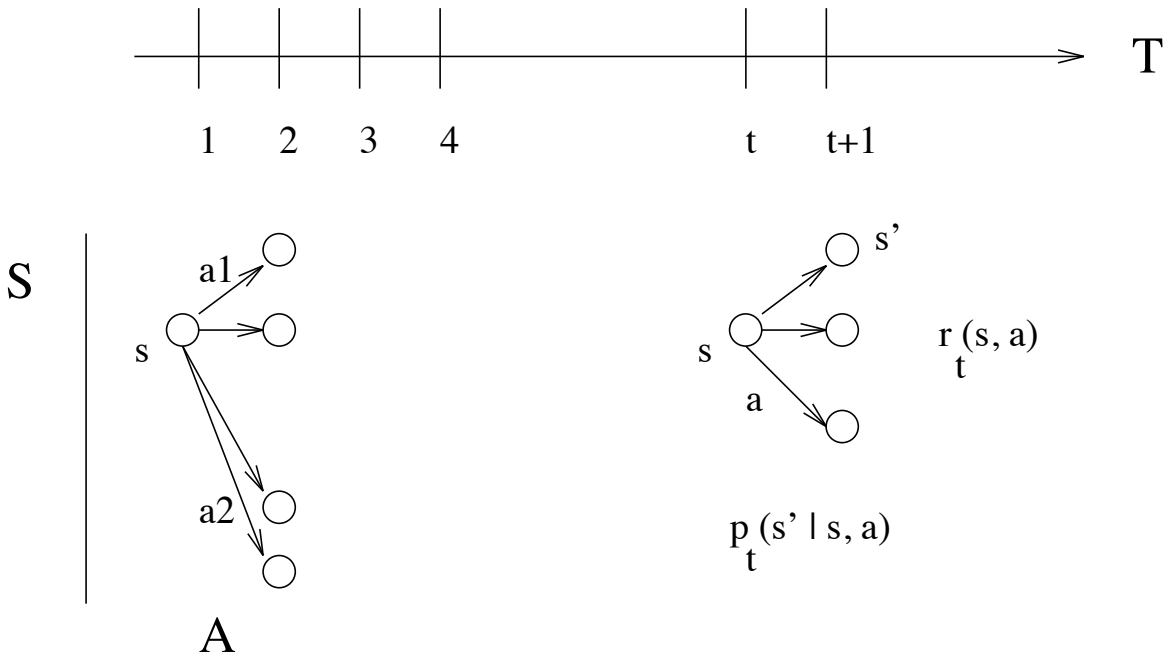


finite-horizon MDP definition

We define a finite-horizon MDP as $< S, A, P, R, h, T >$:

- a set of states S
- a set of actions A
- transition probabilities $P = \{P_a,\ a \in A\}$
- transition rewards $R = \{R_a,\ a \in A\}$
- a terminal reward function h
- a problem horizon T

Markov decision processes



Backward value iteration

Markov property and reward additivity make possible calculus decomposition with a dynamic programming approach
For a general $< S, A, P, R, h, T >$ MDP

$$\begin{aligned} \forall s \in S \quad V_0^*(s) &= h_s \\ \forall s \in S \quad V_t^*(s) &= \max_{a \in A} \left\{ \sum_{s' \in S} P(s' \mid s, a)(r(s, a, s') + V_{t-1}^*(s')) \right\} \end{aligned}$$

for $t = 1, ..., T$.

Computational complexity in $O(TAS^2)$

Backward value iteration with average rewards

Defining

$$r(s,a) = \sum_{s' \in S} P(s' \mid s,a) r(s,a,s'),$$

we've got :

$$\begin{aligned} \forall s \in S \quad V_0^*(s) &= h_s \\ \forall s \in S \quad V_t^*(s) &= \max_{a \in A} \left\{ r(s,a) + \sum_{s' \in S} P(s' \mid s,a) V_{t-1}^*(s') \right\} \end{aligned}$$

for $t = 1, ..., T$.

Finite-horizon optimal policies

The optimal solution is a sequence of policies π_t^*, $t = 1, ..., T$, which are decision rules from S to A :

$$\forall s \in S \quad \pi_t^*(s) \in \underset{a \in A}{\operatorname{argmax}} \left\{ \sum_{s' \in S} r(s,a) + P(s' \mid s,a) V_{T-t}^*(s') \right\}$$

The optimal policy is

- Markov (only depends on the current state)
- deterministic (always the same action in the same state)
- non-stationay (the best action depends on the decision stage t)

Policy families

policy π_t	deterministic	stochastic
Markov	$s_t \longrightarrow a_t$	$a_t,\ s_t \longrightarrow [0,1]$
History-dependent	$h_t \longrightarrow a_t$	$h_t,\ s_t \longrightarrow [0,1]$

where $h_t = (s_1, a_1, s_2, \ldots, s_{t-1}, a_{t-1}, s_t)$
Stationary policies are such that $\forall t\, \pi_t = \pi$

Value functions and finite-horizon criterion

The optimal policy π^* calculated with backward value iteration maximizes the $T-$step value function V_T^π, from S to $\mathbb{R}$:

$$V_T^\pi(s) = E\left[\sum_{t=1}^{t=T} r(s_t, a_t, s_{t+1}) \mid s_1 = s,\ \pi_t\right]$$

$V_T^\pi(s)$ is the expected value of the the sum of rewards when starting in s and following the policy π during T steps

Maximizing value functions

Natural partial order on value functions $V_T^\pi \in \mathbb{R}^S$:

$$\forall U, V \in \mathbb{R}^S \quad U \preceq V \Leftrightarrow \forall s \in S \quad U(s) \leq V(s).$$

But the optimal solution π^* is the same for any initial state s at time $t = 1$:

$$\forall \pi \in \Pi \quad \forall s \in S \quad V_T^\pi(s) \leq V_T^{\pi^*}(s)$$

One can write $V_T^* = \max_{\pi \in \Pi} V_T^\pi = V_T^{\pi^*}$

Computing value functions with backward iteration

For a given policy $\pi = (\pi_1, \ldots, \pi_T)$, its value function V_T^π can be calculated by :

$$\begin{aligned} \forall s \in S \quad V_0^\pi(s) &= h_s \\ \forall s \in S \quad V_t^\pi(s) &= \sum_{s' \in S} r(s, \pi_{T-t+1}(s)) + P(s' \mid s, \pi_{T-t+1}(s)) V_{t-1}^\pi(s') \end{aligned}$$

for $t = 1, ..., T$.

Turnpike planning horizon theorem

Optimal policy π^* and optimal value function V_T^* are non-stationary.

However, there exists some H such that $\forall T > H$, $\pi^*_{1/T} = \pi^*_{1/H}$

For a given MDP problem, the optimal decision at the first stage will be same for any longer planning horizon than the Turnpike planning horizon.

MDP with infinite horizon

Why infinite horizon ?

- a good approximation for large planning horizon
- leads to stationary optimal solutions (cf. turnpike theorem)
- efficient algorithms

Optimality criteria with infinite horizon

For a given policy π one can consider :
The total reward criterion

$$V^{\pi}(s) = E\left[\sum_{t=1}^{\infty} r(s_t, a_t, s_{t+1}) \mid s_1 = s,\ \pi\right]$$

The average reward criterion

$$\rho^{\pi}(s) = \lim_{n\to\infty} E\left[\frac{1}{n}\sum_{t=1}^{n} r(s_t, a_t, s_{t+1}) \mid s_1 = s,\ \pi\right]$$

The γ-discounted criterion

$$V_{\gamma}^{\pi}(s) = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \mid s_1 = s,\ \pi_t\right]$$

Optimality criteria with infinite horizon

Total reward criterion can be used with finite unbounded horizon, like optimal stopping problems. Otherwise the existence of $V^{\pi}(s)$ is not always guaranteed

Average reward criterion reflects the average value of the rewards per step along the trajectory, and is used in many cyclical tasks like queueing control or communication network problems.

The theoretical analysis of these two criteria is more complex than for the discounted criterion, although there exist some efficient corresponding optimization algorithms

The γ-discounted criterion

The most commonly used criterion in infinite horizon

For an MDP $< S, A, P, R, \gamma >$ we look for π^* such that :

$$\forall \pi \quad \forall s \in S \quad V_\gamma^\pi(s) \leq V_\gamma^{\pi^*}(s)$$

with

$$V_\gamma^\pi(s) = E\left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t, s_{t+1}) \mid s_1 = s,\ \pi_t\right]$$

This value function always exists for $0 \leq \gamma < 1$

The discount factor γ

The factor γ can be interpreted as

- the value at time t of a unit reward perceived at time $t+1$. In economy, $\gamma = \frac{1}{1+r}$, where r is the discount rate, or interest rate ;
- the subjective probability that the decision problem will end before the next period

One assumes that decision stages are regularly dispatched on the temporal axis

The discount factor is part of the decision model

Computing the value function V_γ^π

For a given stationary Markov policy π, the discounted value function V_γ^π is unique solution of the following fixed-point equation :

$$\forall s \in S \quad V_\gamma^\pi(s) = \sum_{s' \in S} P(s' \mid s, \pi(s))(r(s, \pi(s), s') + \gamma V_\gamma^\pi(s'))$$

or, with $r(s, a)$ rewards :

$$\forall s \in S \quad V_\gamma^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi(s)) V_\gamma^\pi(s')$$

With vector and matrix notations

We denote by P_π the matrix of the $P(s' \mid s, \pi(s))$ probabilities, and by R_π the vector of the $r(s, \pi(s))$ rewards. P_π and R_π can be built line after line from π and the matrices P_a and R_a
The fixed-point equation becomes :

$$V_\gamma^\pi = R_\pi + \gamma P_\pi V_\gamma^\pi$$

Its solution is given by

$$V_\gamma^\pi = (I - \gamma P_\pi)^{-1} R_\pi$$

$(I - \gamma P_\pi)^{-1}$ exists for $\gamma < 1$ because P_π is a probability matrix.

A fixed-point iterative approach

An Iterative approach :

$$V_{n+1} = R_\pi + \gamma P_\pi V_n,$$

or equivalently

$$\forall s \in S \quad V_{n+1}(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} P(s' \mid s, \pi(s)) V_n(s')$$

until $\| V_{n+1} - V_n \| < \epsilon$

V_n converges toward V_γ^π, $\forall V_0 \in \mathbb{R}$

Bellman's optimality equation

Recall the backward iteration algorithm in finite horizon :

$$\begin{aligned} \forall s \in S \quad V_0^*(s) &= h_s \\ \forall s \in S \quad V_t^*(s) &= \max_{a \in A} \left\{ \sum_{s' \in S} P(s' \mid s, a)(r(s, a, s') + V_{t-1}^*(s')) \right\} \end{aligned}$$

In infinite horizon, we've got at the limit :

$$\forall s \in S \quad V_\infty^*(s) = \max_{a \in A} \left\{ \sum_{s' \in S} P(s' \mid s, a)(r(s, a, s') + \gamma V_\infty^*(s')) \right\}$$

This is the Bellman's equation, with its unique solution $V_\gamma^* = V_\infty^*$

Solving the Bellman's optimality equation

3 main approaches :

- value iteration
- policy iteration
- linear programming

value iteration

An Iterative approach on value function

$$V_{n+1} = \max_a \{R_a + \gamma P_a V_n\},$$

or equivalently

$$\forall s \in S \quad V_{n+1}(s) = \max_a \{r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s, a) V_n(s')\}$$

until $\|V_{n+1} - V_n\| < \epsilon$

V_n converges toward V_γ^*, $\forall V_0 \in \mathbb{R}$
π_n converges toward π^*, $\forall V_0 \in \mathbb{R}$

policy iteration

An iterative approach on policies

$$\pi_{n+1} = \underset{a}{\text{argmax}}\{R_a + \gamma P_a V^{\pi_n}\},$$

until $\pi_{n+1} = \pi_n$

fast convergence toward π^*, $\forall pi_0$

linear programming

Solve :

$$\min_V \sum_{s \in S} V(s)$$

with :

$$V(s) \geq r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s'), \quad \forall s \in S,\ a \in A$$

Polynomial in S, A

Value function approximation

We have seen exact methods where optimal value functions are exactly computed.
In practice, for large-size problems, the perfect representation of these functions is impossible.

We need to approximate these functions (Bertsekas and Tsitsiklis 1996)

Value function approximation

Important result :
V from S to IR, and π a policy "greedy" w.r.t. V :

$$\forall s \in S \quad \pi(s) = \underset{a}{\text{argmax}}\{r(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) V(s')\}$$

Then

$$\| V_\gamma^* - V_\gamma^\pi \| \leq \frac{2\gamma}{1-\gamma} \| V - V_\gamma^* \|$$

This justifies the search for a good approximation of the optimal value function

Value function approximation

- linear approximation with features

$$V_\xi(s) = \xi(1)\psi_1(s) + \cdots \xi(K)\psi_K(s)$$

  where $\psi_i()$ are K features from S to $\mathbb{R}$
- non linear approximation, neural networks

Value function approximation

Approximate value iteration algorithm :

$$V_{n+1} = \mathcal{A}\left(\max_a\{R_a + \gamma P_a V_n\}\right),$$

until $\|V_{n+1} - V_n\| < \epsilon$,
where $\mathcal{A}$ is a projection operator on the set of approximate value functions

Approximate policy iteration algorithm :
- Approximation : compute V_n an approximation of $V_\gamma^{\pi_n}$
- Improvement :

$$\pi_{n+1} = \operatorname*{argmax}_a\{R_a + \gamma P_a V_n\},$$

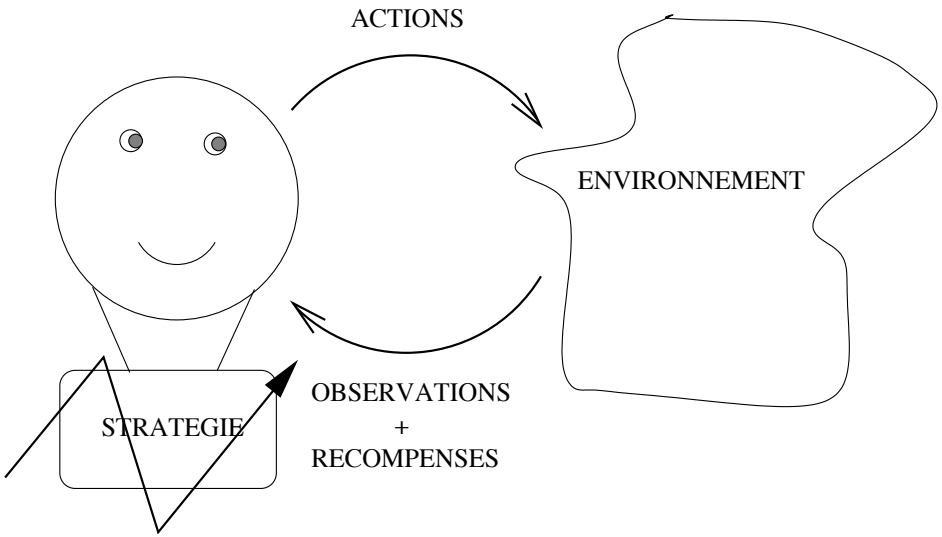
One can show that

$$\lim_{n\to\infty} \|V_\gamma^* - V_\gamma^{\pi_n}\| \leq \frac{2\gamma}{(1-\gamma)^2} \lim_{n\to\infty} \|V_n - V_\gamma^{\pi_n}\|$$

Plan

Reinforcement Learning ...

An autonomous agent exploring his environment, in order to learn a behavior that maximizes the rewards he receives.



What's new ?

Reinforcement Learning goes beyond dynamic programming in two ways :

- learning an optimal policy from experience
$$(\text{state}_n, \text{action}_n, \text{state}_{n+1}, \text{reward}_n)$$
- solving decision problems with large dimensions by **parameterizing** value functions or policies.

(Sutton and Barto 1998)

3 types of machine learning

Supervised learning

- learning from a teacher
- with labelled exemples
- generalizing

Unsupervised learning

- learning from similarities
- with unlabelled exemples
- finding structure

Reinforcement learning

- learning by interaction
- with trial and error
- optimizing behavior

Real-Time Dynamic Programming

RTDP algorithm : learning the optimal value function with *value iteration*, with update in the current state :
After each transition,

$$(s_n,\ a_n,\ s_{n+1},\ r_n)$$

$$\begin{aligned} V_{n+1} &\leftarrow V_n \\ V_{n+1}(s_n) &\leftarrow \max_a \{r(s_n, a) + \gamma \sum_{s'} p(s' \mid s_n, a) V_n(s')\} \end{aligned}$$

Choice of transitions

Current state s_n is the resulting state of the last transition (real-time constraint)
The choice of a_n is directed by the current value function V_n :

$$a_n = \underset{a}{\text{argmax}}\{r(s_n, a) + \gamma \sum_{s'} p(s' \mid s_n, a) V_n(s')\}$$

Optimistic algorithm

Adaptive RTDP

The MDP model is estimated on-line

$$\begin{aligned} p(s'|s,a) &\leftarrow \frac{n^{s'}_{s,a}}{n_{s,a}} \\ r(s,a) &\leftarrow r_n \end{aligned}$$

and a_n visits A (**exploration**)

Q-learning

(Watkins 1999)

No use of the MDP model
Learning of the optimal value function following *value iteration* principle

The Q-value function

Without model, V^* is not sufficient to define the corresponding optimal policy

$$\pi^*(s) = \underset{a}{\text{argmax}}\{r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V^*(s')\}$$

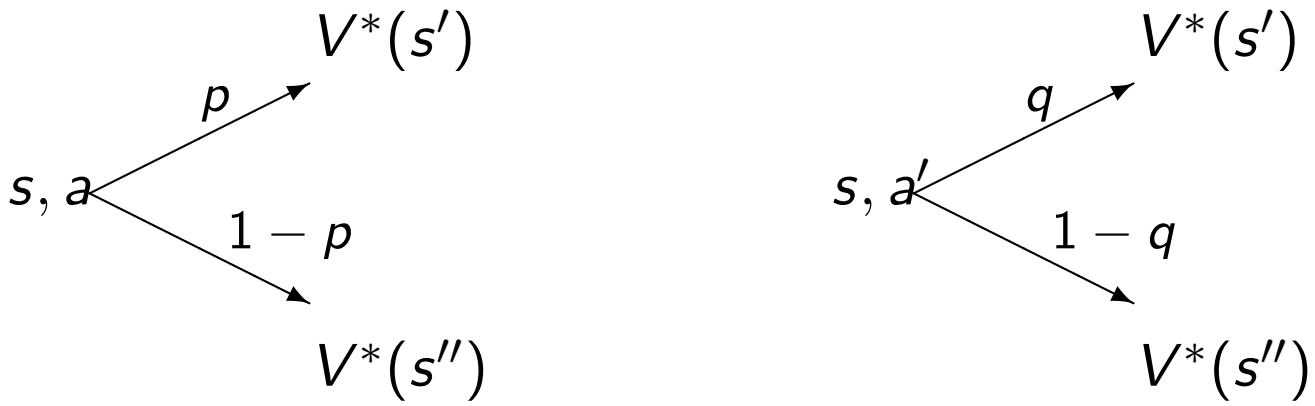
We thus try to learn the Q-value function :

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V^*(s')$$

We have

$$\pi^*(s) = \underset{a}{\text{argmax}}\, Q^*(s,a) \text{ et } V^*(s) = \max_a Q^*(s,a)$$

The Q-value function



$$
\begin{aligned}
Q^*(s,a) &= r(s,\ a) + \gamma\{pV^*(s') + (1-p)V^*(s'')\} \\
Q^*(s,a') &= r(s,\ a') + \gamma\{qV^*(s') + (1-q)V^*(s'')\} \\
\pi^*(s) &= \underset{a,a'}{\text{argmax}}\{Q^*(s,a),\ Q^*(s,a')\}
\end{aligned}
$$

Q-Learning principle

Value iteration

$$
\begin{aligned}
V_{n+1}(s) &\leftarrow \max_a \overbrace{\{r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V_n(s')\}}^{Q_n(s,a)} \\
\Rightarrow \quad Q_{n+1}(s,a) &= r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) V_{n+1}(s')\} \\
Q_{n+1}(s,a) &\leftarrow r(s,a) + \gamma \sum_{s'} p(s' \mid s,a) \max_{a'} Q_n(s',a')
\end{aligned}
$$

For the pair (s_n, a_n), we estimate the sum by

$$r_n + \gamma \max_{a'} Q_n(s_{n+1}, a')$$

Q-learning algorithm

After each transition,

$$(s_n,\ a_n,\ s_{n+1},\ r_n)$$

$$\begin{aligned} Q_{n+1} &\leftarrow Q_n \\ \delta_n &\leftarrow r_n + \gamma \max_{a'} Q_n(s_{n+1}, a') - Q_n(s_n, a_n) \\ Q_{n+1}(s_n, a_n) &\leftarrow Q_n(s_n, a_n) + \alpha_n \delta_n \end{aligned}$$

with $\lim_{n\to\infty} \alpha_n = 0$ (e.g. $\frac{1}{n}$ ou $\frac{1}{n_{s,a}}$)

Stochastic approximation and Q-learning

Bellman equation :

$$V^*(s) = \max_a \sum_{s'} p(s' \mid s, a)\{r(s, a, s') + \gamma V^*(s')\}, \quad \forall s$$

Equivalently,

$$\begin{aligned} V^*(s) &= \max_a Q^*(s, a), \quad \forall s \\ Q^*(s, a) &= \sum_{s'} p(s' \mid s, a)\{r(s, a, s') + \gamma \max_{a'} Q^*(s', a')\}, \quad \forall s, a \\ Q^*(s, a) &= E[r(s, a, s') + \gamma \max_{a'} Q^*(s', a')], \quad \forall s, a \end{aligned}$$

Stochastic approximation and Q-learning

Let $\xi = (s, a, r, s')$, and $H(Q, \xi)_{s,a} = (r + \gamma \max_{a'} Q(s', a') - Q(s, a))$
Then

$$E[H(Q^*, \xi)] = 0$$

The Robbins-Monro algorithm gives directly

$$Q_{n+1} = Q_n + \alpha_n H(Q_n, \xi_n)$$

that is

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n(r_n + \gamma \max_{a'} Q_n(s_{n+1}, a') - Q_n(s_n, a_n))$$

Q-learning convergence

For a good decrease rate of α_n to 0 (e.g. $\frac{1}{n}$ ou $\frac{1}{n_{s,a}}$), and if all state-action pairs of $S \times A$ are infinitely visited, then Q-learning converges to Q^* with probability $= 1$.

The Exploration / Exploitation dilemna

Le choix de l'état est souvent lié à la dynamique.
Pour l'action, une démarche optimiste consiste à choisir à chaque itération la meilleure action courante

$$a_n = \underset{a}{\operatorname{argmax}}\, Q_n(s_n, a)$$

Plus raisonnablement, il convient de régulièrement choisir une action aléatoire dans A.
On utilise ainsi des **fonctions d'exploration dirigées** ou **non-dirigées**

Fonctions d'exploration non-dirigées

- suivre Q_n pendant N_1 transitions, puis tirer uniformément dans A pendant N_2 transitions
- à chaque itération suivre Q_n avec une probabilité τ, ou tirer uniformément dans A avec une probabilité $1 - \tau$
- tirer a_n dans A selon

$$p_T(a) = \frac{e^{-\frac{Q_n(s_n,a)}{T}}}{\sum_{a'} e^{-\frac{Q_n(s_n,a')}{T}}}$$

$$\text{avec } T \underset{\infty}{\rightarrow} 0$$

Guided exploration policies

On utilise l'information accumulée au cours de la recherche, autre que Q_n.
En pratique, on ajoute à Q_n un **bonus d'exploration** et on choisit la fonction qui maximise cette somme.
Exemple de bonus

- la *recency based method* : $Q_n(s,a) + \varepsilon\sqrt{(T_{s,a})}$
  $T_{s,a}$ est le nombre d'itérations depuis le dernier choix de a dans s
- la *uncertainty estimation method* : $Q_n(s,a) + \varepsilon/n_{s,a}$
- UCB1 (regret minimization) : $Q_n(s,a) + \sqrt{2log(n)/(n_{s,a}}$

On-Policy and Off-Policy learning rules

Dans le Q-learning, la politique d'exploration intervient juste pour générer les expériences $\xi = (s_n, a_n, r_n, s_{n+1})$ (Off-policy)

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n(r_n + \gamma \max_{a'} Q_n(s_{n+1}, a') - Q_n(s_n, a_n))$$

L'algorithme SARSA adapte la règle d'apprentissage en prenant en compte explicitement la politique d'exploration et l'action a_{n+1} (On-policy) :

$$Q_{n+1}(s_n, a_n) = Q_n(s_n, a_n) + \alpha_n(r_n + \gamma Q_n(s_{n+1}, a_{n+1}) - Q_n(s_n, a_n))$$

Sarsa converge également vers Q^*

Joint learning of a value fuction and a policy

Dans le Q-learning, la politique apprise est directement liée à la fonction Q apprise.
Il est aussi possible de gérer explicitement une suite de politiques π_n et une suite de fonctions de valeur V_n. On suit alors le même principe que dans l'algorithme de *policy iteration*

Principe des *actor-critic methods*

On maintient π_n (*action network*) et V_n (*critic network*)
Après chaque transition (s_n, a_n, s_{n+1}, r_n)

- ($p(s' \mid s, a)$ et $r(s, a)$ sont mis à jour)
- V_n est mise à jour
- π_n est améliorée

Est souvent associé à une représentation paramétrée de V_n et π_n (voir plus loin).

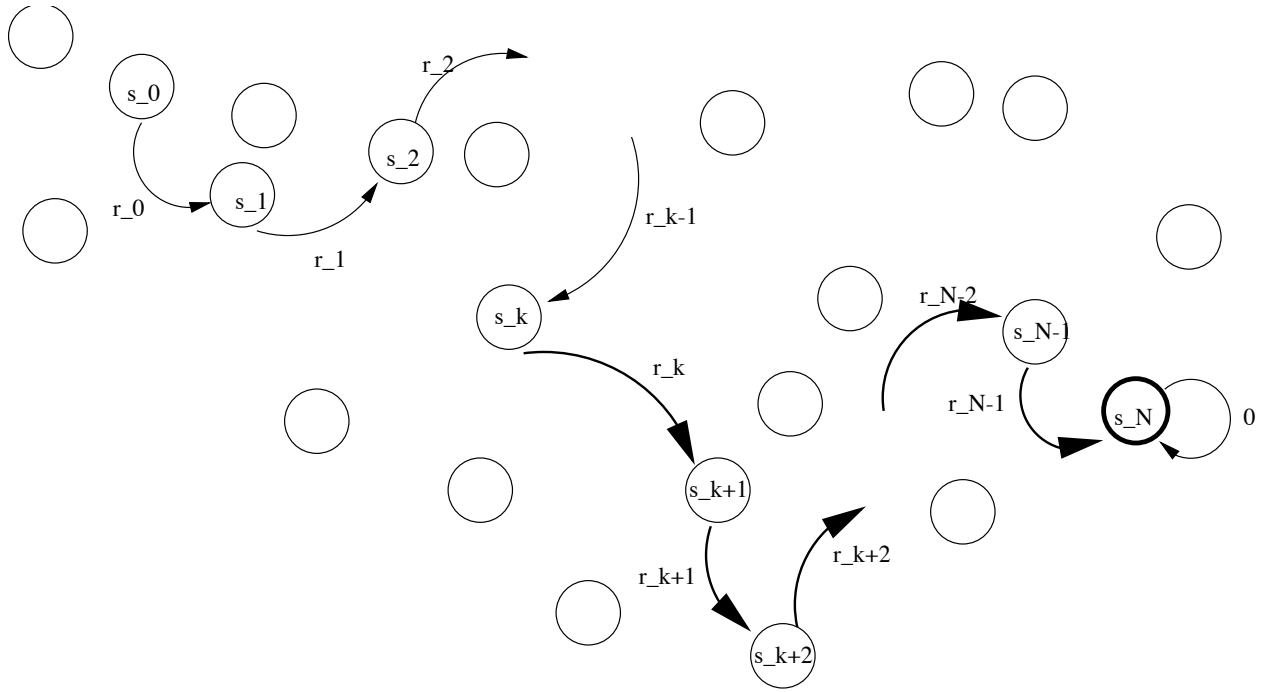
Apprentissage de la fonction de valeur d'une politique

La valeur normale de V_n devrait être V_{π_n}. Différentes méthodes permettent de l'approcher

- *maximum de vraisemblance* : calcul de V_{π_n} sur la base des estimées $p()$ et $r()$; peu efficace
- *Monte Carlo*
- *Programmation Dynamique*
- *TD(λ)*

Méthode de Monte Carlo

Dans le cadre $\gamma = 1$ avec état absorbant
Pour une politique π fixée
On observe $(s_0, \ldots, s_N)$ et $(r_0, \ldots, r_{N-1})$



Algorithmes *off-line* / *on-line*

A la fin de chaque trajectoire, on met à jour les N valeurs $V(s_k)$ par

$$V(s_k) \leftarrow V(s_k) + \alpha(r_k + r_{k+1} + \cdots + r_{N-1} - V(s_k))$$

On peut aussi modifier V après chaque transition (s_k, s_{k+1}, r_k), en utilisant les **différences temporelles** :

$$r_k + r_{k+1} + \cdots + r_{N-1} - V(s_k) = d_k + d_{k+1} + \cdots + d_{N-1}$$

avec $\boxed{d_k = r_k + V(s_{k+1}) - V(s_k)}$

$$V(s_l) \leftarrow V(s_l) + \alpha d_k, \quad l = 0, \ldots, k$$

La méthode de la programmation dynamique

On peut aussi chercher à résoudre stochastiquement le système d'équations linéaires définissant V :

$$V(s) = r(s, \pi(s)) + \sum_{s'} p(s' \mid s, \pi(s)) V(s')\}$$

Après chaque transition (s_k, s_{k+1}, r_k)

$$\begin{aligned} V(s_k) &\leftarrow V(s_k) + \alpha(r_k + V(s_{k+1}) - V(s_k)) \\ &\leftarrow V(s_k) + \alpha d_k \end{aligned}$$

La méthode des différences temporelles (TD(λ))

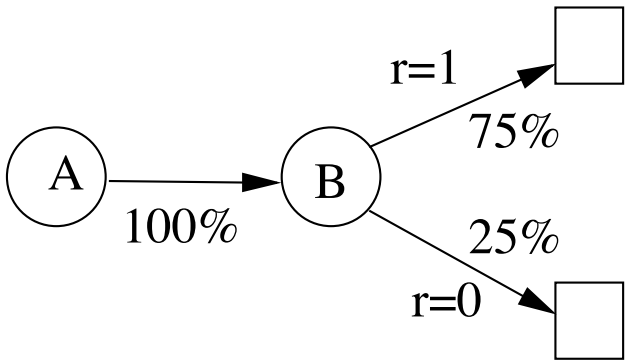
TD(λ) est un compromis entre les deux précédentes méthodes
Pour une trajectoire observée $(s_0, \ldots, s_N)$ et $(r_0, \ldots, r_{N-1})$ la fonction de valeur courante V est mise à jour selon

$$V(s_k) \leftarrow V(s_k) + \alpha \sum_{m=k}^{m=N-1} \lambda^{m-k} d_m, \quad k = 0, \ldots, N-1$$

- $\lambda = 0$: méthode de la programmation dynamique
- $\lambda = 1$: méthode de Monte Carlo

La convergence presque sûre est assurée.

Comparaison entre Monte Carlo et Programmation Dynamique (TD(0))



(A, 0, B, 0)	(B,1)
(B,1)	(B,1)
(B,1)	(B,1)
(B,1)	(B,0)

Que vaut $V(B)$? et $V(A)$?

Comparaison entre Monte Carlo et Programmation Dynamique (TD(0))

- $V(B) \approx \frac{3}{4}$
- $V(A) \approx 0$ pour Monte Carlo
- $V(A) \approx \frac{3}{4}$ pour TD(0)

Monte Carlo minimise l'erreur quadratique sur les observations.
TD(0) maximise la vraisemblance de l'estimation.

TD(λ) *on-line*

Après chaque transition (s_k, s_{k+1}, r_k),

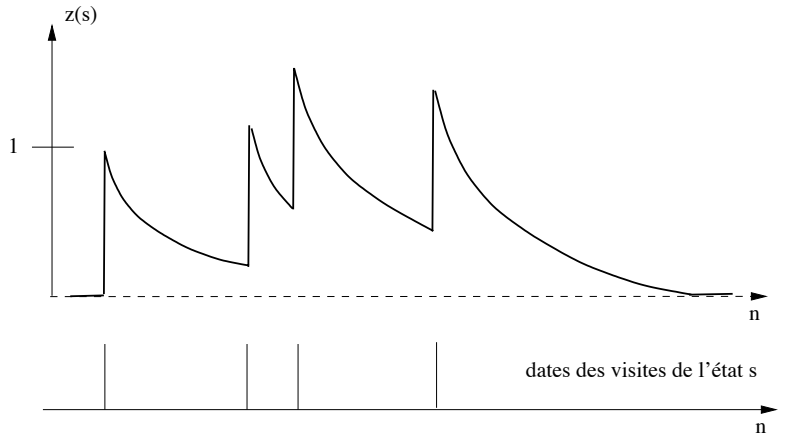
$$V(s_l) \leftarrow V(s_l) + \alpha\lambda^{k-l} d_k, \quad l = 0, \ldots, k$$

On utilise surtout une forme similaire de cet algorithme utilisant la notion de **trace d'éligibilité**.

$$V(s) \leftarrow V(s) + \alpha z_k(s) d_k \quad \forall s \in S$$

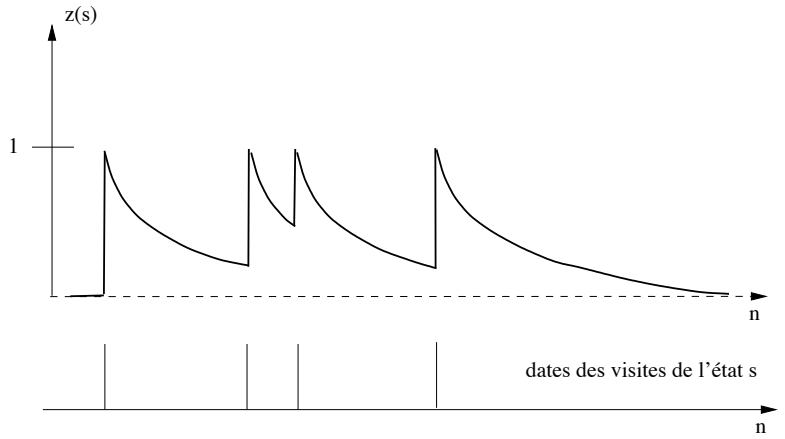
Trace d'éligibilité accumulative

$$
\begin{aligned}
z_0(s) &= 0, \quad \forall s \in S \\
z_n(s) &= \begin{cases} \gamma\lambda z_{n-1}(s) & \text{si } s \neq s_n \\ \gamma\lambda z_{n-1}(s) + 1 & \text{si } s = s_n \end{cases}
\end{aligned}
$$



Trace d'éligibilité avec réinitialisation

$$
\begin{aligned}
z_0(s) &= 0, \quad \forall s \in S \\
z_n(s) &= \begin{cases} \gamma\lambda z_{n-1}(s) & \text{si } s \neq s_n \\ 1 & \text{si } s = s_n \end{cases}
\end{aligned}
$$



TD(λ) et Q-learning

Si TD(λ) est utilisé dans les méthodes de type *policy iteration* pour évaluer une politique, il peut aussi être utilisé pour améliorer le Q-learning
Après chaque expérience, ou transition $(s_n,\ a_n,\ s_{n+1},\ r_n)$

$$
\begin{aligned}
z_n &\leftarrow 0 \text{ si } a_n \neq \underset{a}{\operatorname{argmax}}\, Q_n(s_n, a) \\
z_n(s_n, a_n) &\leftarrow z_n(s_n, a_n) + 1 \\
\delta_n &\leftarrow r_n + \gamma \max_{a'} Q_n(s_{n+1}, a') - Q_n(s_n, a_n) \\
Q_{n+1}(s, a) &\leftarrow Q_n(s, a) + \alpha_n z_n(s, a)\delta_n \\
z_{n+1}(s, a) &\leftarrow \gamma\lambda z_n(s, a)
\end{aligned}
$$

Plan

Représentation de la fonction de valeur

Pour des problèmes discrets de faibles tailles, il est possible de représenter $V = V^{\pi}$ ou $V = V^{*}$ (ou Q) par un tableau état :valeur (**look-up table**).
L'apprentissage de V porte alors directement sur ses composantes.
Pour des problèmes de grande dimension ou à domaines continus, il est nécessaire de passer par une représentation paramètrée

$$\mathbf{V} = \mathbf{V}_{\xi} \text{ ou } \mathbf{Q} = \mathbf{Q}_{\xi}$$

où ξ est un vecteur de paramètres de plus faible taille.
L'apprentissage porte alors sur ξ

$$\xi_{\mathbf{n+1}} = \xi_{\mathbf{n}} + \alpha\Delta(\mathbf{s_n}, \mathbf{a_n}, \mathbf{s_{n+1}}, \mathbf{r_n})$$

Caractéristiques d'une architecture

la capacité d'approximation : on cherche à minimiser l'erreur intrinsèque $\epsilon = \min_\xi \|V - V_\xi\|$;

la capacité de généralisation : on recherche des ξ de faible dimension minimisant $\epsilon \Rightarrow$ les valeurs de plusieurs états seront simultanément modifiées avec la présentation d'un seul exemple ;

la simplicité de l'évaluation : selon la paramétrisation retenue, le calcul pour un état $s \in S$ de $V_\xi(s)$ peut être plus ou moins coûteux ;

l'efficacité de l'apprentissage : selon ξ, la convergence n'est plus toujours assurée, et les algorithmes peuvent devenir très complexes.

Différentes architectures d'approximation

- représentations différentiables
  - linéaires
  - réseaux neuronaux
  - ...
- représentations non différentiables
  - arbres de régression,
  - règles de décision
  - ...

Représentations différentiables

Il est naturel ici d'utiliser des méthodes de type descente de gradient stochastique pour apprendre le vecteur de paramètres ξ

$$\xi_{n+1} = \xi_n + \alpha_n(R_n - V_{\xi_n}(s_n))\nabla_{\xi_n} V_{\xi_n}(s_n)$$

où R_n est l'estimation directe tirée de l'expérience de la valeur de V en s_n (cf. Monte Carlo, TD(0) et TD(λ)).
Dans le cas général, seul Monte Carlo assure que cette règle de mise à jour converge vers un optimum local pour

$$\epsilon(\xi) = \sum_{s \in S}(V(s) - V_\xi(s))^2$$

Les représentations linéaires

$$V_\xi(s) = \xi(1)\psi_1(s) + \cdots \xi(K)\psi_K(s)$$

où les $\psi_i()$ sont K fonctions de S dans $\mathbb{R}$

$$\nabla_{\xi_n} V_{\xi_n}(s_n) = (\psi_1(s_n), \ldots, \psi_K(s_n))^T$$

entraine

$$\xi_{n+1}(i) = \xi_n(i) + \alpha_n(R_n - V_{\xi_n}(s_n))\psi_i(s_n), \quad \forall i = 1, \ldots, K$$

Il existe un optimum unique ξ^* pour l'erreur quadratique, et TD(λ) converge nécessairement (mais non forcément vers ξ^*)

Les fonctions de voisinage

On définit K régions $\mathcal{R}_i$ qui forment ensemble une couverture de S, et on pose pour $i = 1, \ldots, K$ $\psi_i() = \mathbb{1}_{\mathcal{R}_i}()$ la fonction indicatrice de $\mathcal{R}_i$

$$\forall s \in S \quad \psi_i(s) = \begin{cases} 1 & \text{si } s \in \mathcal{R}_i \\ 0 & \text{sinon} \end{cases}$$

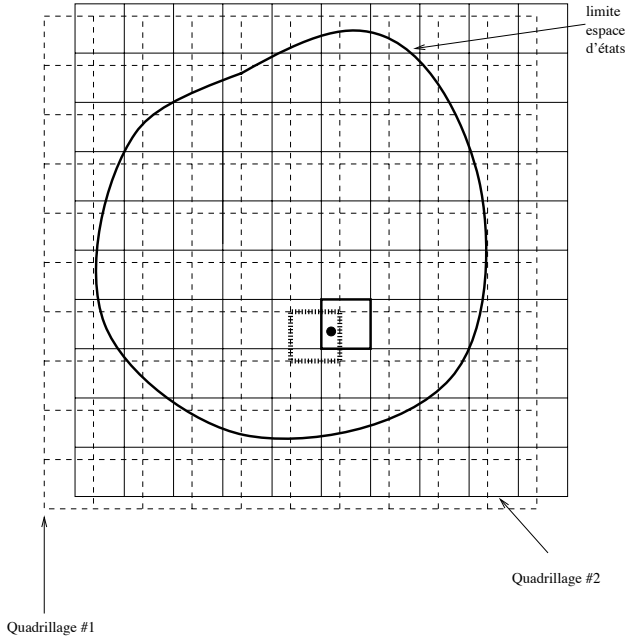
Exemple : *state aggregation*, où les $\mathcal{R}_i$ forment une partition de S.

La méthode des CMAC (*cerebellar model articulator controller*)

Méthode simple utilisée pour des espaces continus de faible dimension

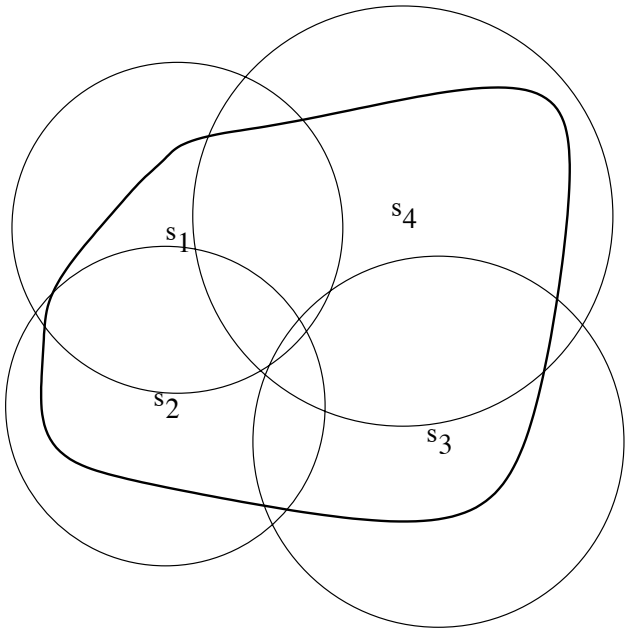
Principe : définir C partitions (ou grilles) de S, décalées géométriquement les unes par rapport aux autres

À chaque région de chaque grille, on associe un poids $\xi(i)$. La valeur d'un état s de S est la somme des C poids des régions de chaque grille qui le contiennent.



Les méthodes à base d'états représentatifs

On place K points s_i dans S, centres d'une région



La forme de la région est fixe, les recouvrements sont possibles

Les méthodes à base d'états représentatifs

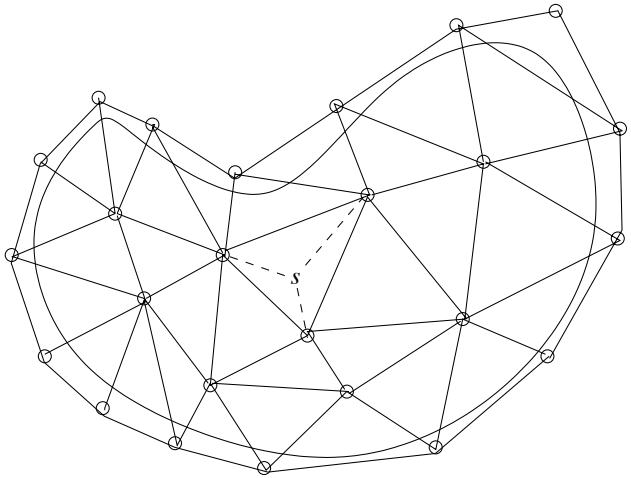
On peut aussi associer une valeur réelle dans l'intervalle $[0, 1]$ fonction de la distance au centre.
Exemple : les *radial basis functions*

$$\forall s \in S \quad \psi_i(s) = \exp(-\frac{\|s - s_i\|}{2{\sigma_i}^2})$$

On peut alors apprendre aussi les meilleurs paramètres s_i et σ_i (architecture non-linéaire)

triangularisation de l'espace d'états

On triangularise $S \subset \mathbb{R}^n$ par les points s_i



On définit la fonction de valeur $V_\xi(s)$ en tout point s comme la combinaison linéaire

$$V_\xi(s) = \lambda_{s_{i_0}}(s) V_\xi(s_{i_0}) + \cdots \lambda_{s_{i_n}}(s) V_\xi(s_{i_n})$$

Les fonctions coordonnées

On projette S dans un sous-espace $\mathbb{R}^K$, en définissant K nouvelles coordonnées fonctions des anciennes.
Les nouvelles coordonnées résument le mieux possible les propriétés d'un état vis-à-vis de la tâche à apprendre.
Exemple : Tétris. $h \times l$ variables d'états binaires $\Rightarrow 2l + 1$ coordonnées

- les hauteurs h_k des l colonnes
- les différences $\mid h_k - h_{k+1} \mid$
- la hauteur maximale du mur $\max_k h_k$
- le nombre de trous dans le mur

Les fonctions heuristiques

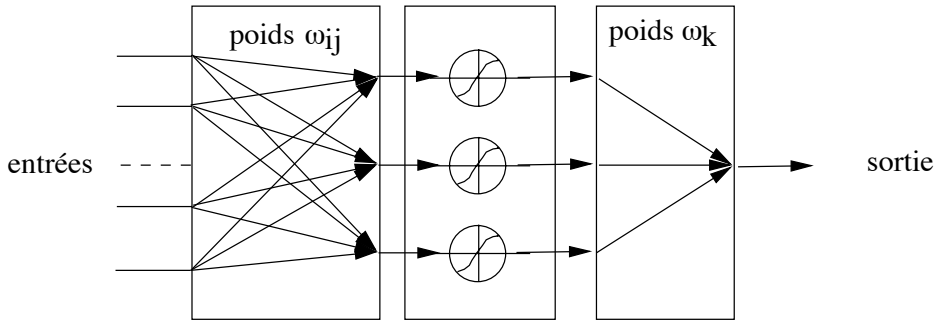
Permet d'exploiter la connaissance de stratégies répondant partiellement au problème
Les $\psi_i()$ sont les fonctions de valeur associées à des politiques obtenues par expertise ou par une première résolution approchée.

$$V_\xi(s) = \xi(1)V_{\pi_1}(s) + \cdots \xi(K)V_{\pi_K}(s)$$

Les V_{π_i} est estimées par simulation, et elles-même paramétrées

Approximations non-linéaires : le perceptron multi-couches

L'architecture non-linéaire aujourd'hui la plus employée en apprentissage par renforcement.



Pour un état $s = (s(1), \ldots, s(i), \ldots)$, $V(s)$ est approchée par

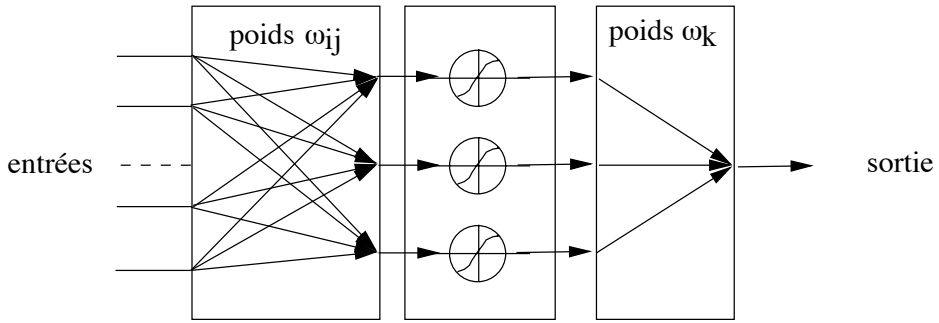
$$V_\xi(s) = \sum_k \omega_k \sigma \left(\sum_i \omega_{k,i} s(i) \right)$$

avec (par exemple)

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Les perceptrons multi-couches sont une généralisation de ce principe avec des réseaux où s'enchaînent couches linéaires et couches sigmoïdales.

Apprentissage du perceptron multi-couches



Le vecteur ξ est constitué des poids ω_i, ω_{jk}, ... du réseau.
L'apprentissage de ces poids est réalisé à partir de la règle de gradient.
Le calcul du terme de gradient $\nabla_{\xi_n} V_{\xi_n}$ est effectué par des techniques de rétro-propagation au sein du réseau.
Il existe de nombreux algorithmes d'apprentissage par renforcement couplant Q-learning et/ou TD(λ) avec un perceptron (cf. deep RL).

Apprentissage de structures non-différentiables

Toutes les structures des représentations précédentes doivent être adaptées à la tâche à apprendre.
Il existe quelques algorithmes d'apprentissage par renforcement cherchant à apprendre une telle résolution spatiale optimale.
Les méthodes de gradient ne sont plus adaptées : gradient tree boosting, algorithmes évolutionnaires, etc.

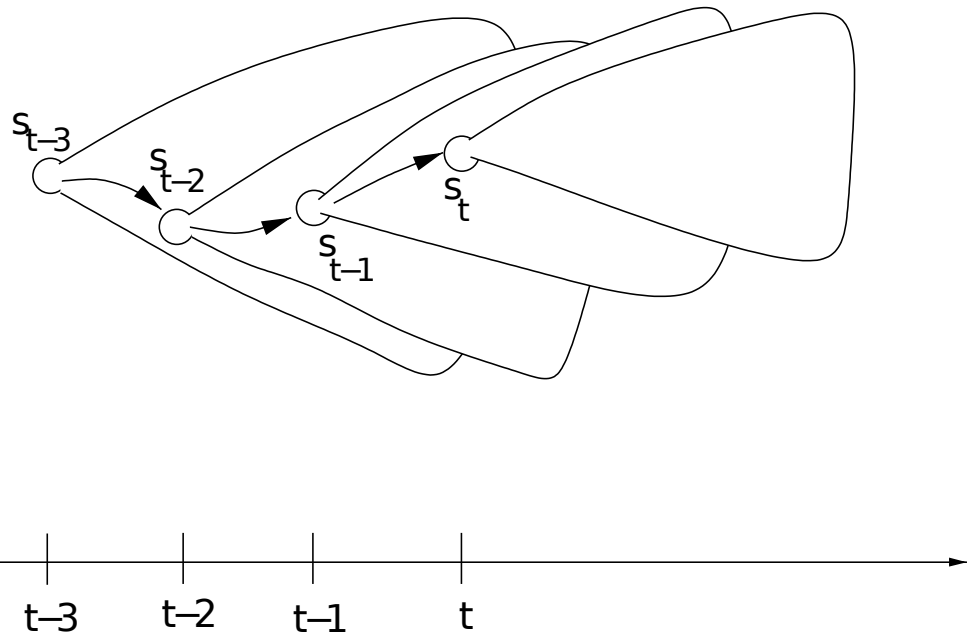
Calcul en ligne des politiques

Pour des problèmes de grandes dimension, le passage exacte en fin ou en cours d'apprentissage de V ou Q vers a ou π peut être très coûteux et doit être approché :

$$\begin{aligned}
\hat{\pi}(s) &= \underset{a}{\operatorname{argmin}}\, Q_\omega(s,a) \\
&= \underset{a}{\operatorname{argmin}} \sum_{s'} \hat{p}(s' \mid s,a)\{r(s,a,s') + \gamma V_\omega(s')\} \\
&= \underset{a}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \{r(s,a,s'_i) + \gamma V_\omega(s'_i)\}
\end{aligned}$$

Online search by simulation

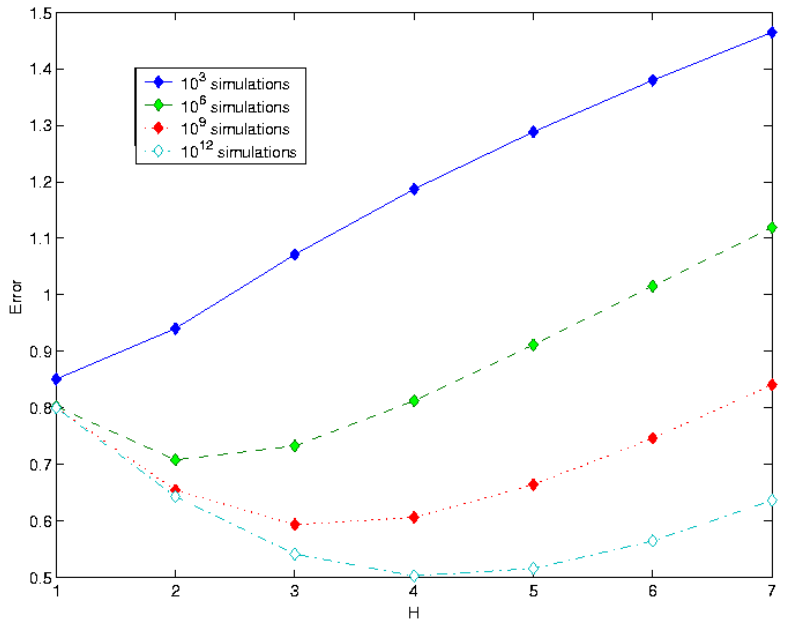
On peut également approcher π uniquement pour l'état courant, sur un horizon $H > 1$ (Roll-out)



On-line resolution techniques

Iterative allocation of simulations

Possible pathology of the simulation-based forward search



Policy approximation

Il est toutefois souvent intéressant d'approximer

$$\pi_n(s) = \underset{a}{\text{argmax}}\, Q_n(s, a)$$

par une fonction paramètrée (*action network*)

$$\pi_n = \pi_{\theta_n}$$

La dynamique de θ_n peut être alors être plus ou moins entrelacée à celle de V_n ou Q_n (actor-critic architecture).

Direct policy search

Policy parameterization $\pi_\theta(s)$, with $\theta \in \Theta \subseteq \mathbb{R}^p$

- we look for optimal parameters θ^* that optimize the expected performance

$$J(\theta^*) = \max_{\theta \in \Theta} V_\gamma^{\pi_\theta},$$

- based on observed trajectories following π_θ obtained by simulation

Policy approximation

We typically consider stochastic policies, like :

$$\pi_\theta(s) = a \text{ with probability } q(a \mid s; \theta) \quad \forall s$$

For instance :

$$q(a \mid s; \theta) = \frac{\theta_{s,a}}{\sum_b \theta_{s,b}}, \quad , \theta_{s,a} \geq 0$$

$$q(a \mid s; \theta) = \frac{\exp \theta.\phi(s,a)}{\sum_b \exp \theta.\phi(s,b)}$$

with

$$\phi(s,a) = (\phi_1(s,a), \ldots, \phi_p(s,a))$$

Policy gradient and one-step MDP

One step MDP ($T = 1$), random initial state following $\mu(s)$

$$J(\theta^*) = \max_\theta E[r(s, a, s')]$$

Score function method

$$\begin{aligned}
\nabla E[r(s,a,s')] &= \nabla \left[\sum_{s,a,s'} \mu(s) q(a|s;\theta) p(s'|s,a) r(s,a,s') \right] \\
&= \sum_{s,a,s'} \left[\frac{\nabla q(a|s;\theta)}{q(a|s;\theta)} r(s,a,s') \right] \mu(s) q(a|s;\vec{\theta}) p(s'|s,a) \\
&= E\left[\frac{\nabla q(a|s;\theta)}{q(a|s;\theta)} r(s,a,s') \right].
\end{aligned}$$

Policy gradient and one-step MDP

With $q(a \mid s;\theta) = \frac{\exp \theta.\phi(s,a)}{\sum_b \exp \theta.\phi(s,b)}$:

$$\frac{1}{q(a|s;\theta)} \frac{\partial q(a|s;\theta)}{\partial \theta_i} = \phi_i(s,a) - \sum_b \phi_i(s,b) q(b|s;\theta),$$

thus

$$\frac{\nabla q(a|s;\theta)}{q(a|s;\theta)} = \phi(s,a) - \sum_b q(b|s;\theta)\phi(s,b)$$

Policy gradient and MDP

Horizon $T \geq 1$

$$J(\theta^*) = \max_\theta E[r(s_0, a_0, s_1) + \gamma r(s_1, a_1, s_2) + \cdots + \gamma^{T-1} r(s_{T-1}, a_{T-1}, s_T)]$$

A similar approach leads to

$$\nabla J(\theta) = \nabla E[\sum_{t=0}^{T-1} \gamma^t r_t] = E\left[\sum_{t=0}^{T-1} \gamma^t r_t \sum_{t'=0}^{t-1} \frac{\nabla q(a_{t'}|s_{t'};\theta)}{q(a_{t'}|s_{t'};\theta)}\right]$$

Make possible sampling approaches and online methods with trace eligibilities

$$z_t = \sum_{t'=0}^{t-1} \frac{\nabla q(a_{t'}|s_{t'};\theta)}{q(a_{t'}|s_{t'};\theta)}$$

Many policy gradient algorithms !

Policy gradient and MDP

Infinite horizon, Bellman equation :

$$V_\theta(s) = \sum_a q(a|s;\theta) \sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V_\theta(s'))$$

Then we have

$$\nabla V_\theta(s) = E\left[\sum_{t=0}^{\infty} \gamma^t \frac{\nabla q(a_t|s_t;\theta)}{q(a_t|s_t;\theta)} Q_\theta(s,a)|s_0 = s\right]$$

thus

$$\begin{aligned} \nabla J(\theta) &= \sum_s \mu(s) \nabla V_\theta(s) \\ &= \sum_s \mu_\theta^\gamma(s) \sum_a \nabla q(a|s;\theta) Q_\theta(s,a) \end{aligned}$$

Actor-critic methods

Use of an approximate value function $Q_\omega(s, a)$, estimated during simulation

Compatibilty criterion between parameters ω and θ

$$\nabla Q_\omega(s, a) = \frac{\nabla q(a|s; \theta)}{q(a|s; \theta)}$$

For instance with $q(a \mid s; \theta) = \frac{\exp \theta.\phi(s,a)}{\sum_b \exp \theta.\phi(s,b)}$:

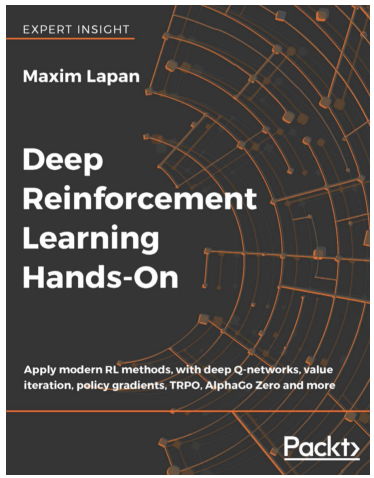
$$Q_\omega(s, a) = \omega.(\phi(s, a) - \sum_b q(b|s; \theta)\phi(s, b))$$

Q_ω linéaire en ϕ_i

Deep Reinforcement Learning

Outperforms all previous state-of-the-art methods

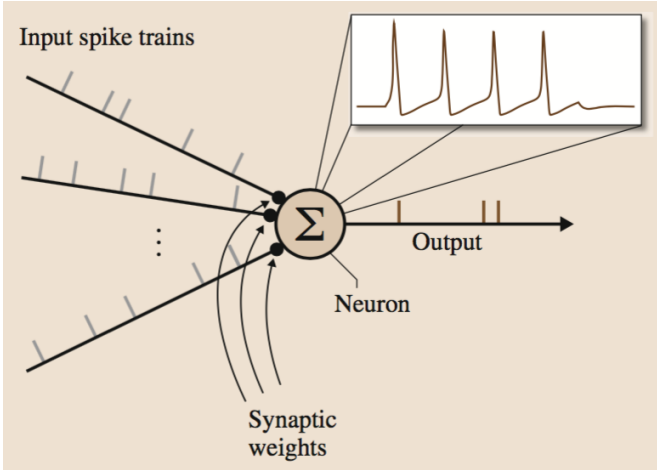
- neural networks to estimate Q_ω or π_θ
- unstable : target network, experience replay, asynchronous methods
- deep Q-learning (DQN, ...)
- deep policy gradient and actor-critic methods (DDPG, A3C, ACER, ...)



Plan

Spiking neurons

- next generation of neural networks
- biologically realistic models of neurons
- discrete events rather than continuous values
- many neuronal models : Hodgkin-Huxley (micro), Leaky Integrate-and-Fire, Spike Response Model, Thorpe Model (macro), etc.



Leaky integrate-and-fire neurons

Voltage membrane potential v, excitatory input current I

$$\tau_m \frac{dv(t)}{dt} = v_{rest} - v(t) + R.I(t)$$

When v reaches a threshold v_{th}, the neuron fires a spike and v is reset to v_{rest}.
A synaptic model of the excitatory input current I of a neuron :

$$I(t) = \sum_j w_j \sum_f \epsilon(t - t_j^f)$$

where w_j is the connection strength of the synapse from neuron j, t_j^f the firing times and $\epsilon()$ the time response of the spike.

Learning in spiking neural networks

Like for traditional neural networks, and more generally for learning systems, 3 different learning paradigms :

- supervised learning
- unsupervised learning
- reinforcement learning

Explicit time dependence, asynchronous information processing, make learning procedures more complex than for the classical perceptron approach.
Promising results for unsupervised and supervised deep learning with spiking neural networks

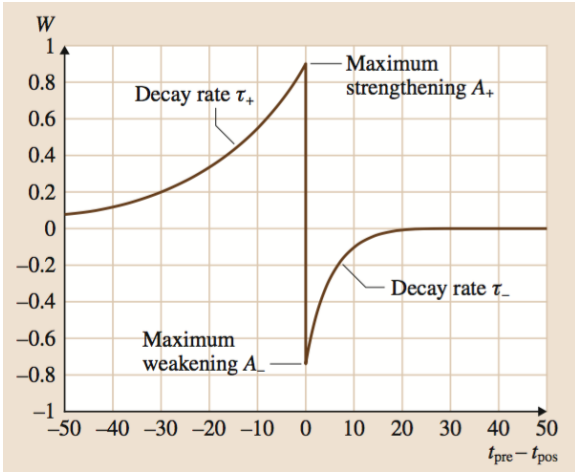
Hebbian learning vs backpropagation

"*Hebbian rule : when an axon of cell A persistently takes part in firing cell B, some growth process or metabolic change takes place to increase A's efficacy as one of the cells firing B.*" - Donald Hebb, 1949

- Hebbian learning corresponds to an unsupervised setting : no need of external target
- supervised learning relies typically of gradient based descent, to modify synaptic weights with back propagation in order to minimize the error between the network output and the desired target.

STDP- Spike-Timing Dependent Plasticity

- directly inspired by Hebb's law.
- synaptic changes dependent on the relative timing of pre and post synaptic spikes
- with a learning window



$$\Delta w_j = \sum_f \sum_n W(t^n - t_j^f)$$

- online implementation with traces

$$\tau_+ \frac{dz_j}{dt} = -z_j + A_+ \sum_f \delta(t - t_j^f) \text{ and } \tau_- \frac{dz}{dt} = -z + A_- \sum_n \delta(t - t^n)$$

- many variants of STDP models

STDP and Reinforcement Learning

How to model synaptic changes considering a global reward signal ?
Few propositions in the litterature for RL and spiking neuron networks, more or less related to STDP (Gerstner 2000, Seung 2003, Xi & Seung 2004, Florian 2007, Takita & Hagiwara 2005, El-Laithy & Bogdan 2011, etc.)
The difficulties are :

- how to take into account the credit assignment problem ?
- how to obtain a local biological plausible learning rule ?

Modulated STDP, with eligibility traces for hebbian learning and additional "gating signals" like rewards, leads to "neo-hebbian three-factor" learning rules (Gerstner at al. 2018)

$$\Delta w_{ij} = \alpha z_{ij} r$$

Back to MDP and RL frameworks

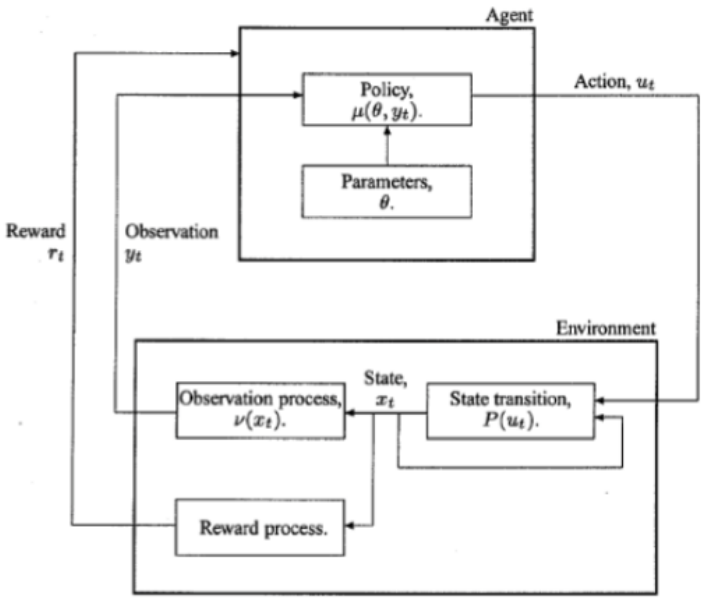
With STDP reinforcement learning, we want to learn the optimal synaptic strengths w_{ij} when every neuron is an agent, and there is a global reward r_t for the whole network

The classical MDP and RL frameworks must be extended, e.g. :

- TMDP, SMDP, GSMDP for time-dependence
- DEC-MDP for decentralized decison-making
- GMDP for graph-based representation
- POMDP for partially observable states

POMDP framework

- MDP for state (x) - action (u) probabilistic dynamics
- reward $r(x)$
- observation distribution $\nu_y(x)$ prob. of observing y in state x
- stochastic policy $\mu_u(y)$ prob. of choosing action u for observation y



- Parameterized policy $\mu_u(y) = \mu(\theta, \ldots)$
- we want to maximize $J(\theta) = \lim_{n\to\infty} E_\theta \left[\frac{1}{n}\sum_{t=1}^{n} r(s_t)\right]$

Direct RL approach (OLPOMDP, Baxter & Bartlett)

- the trace

$$z_{t+1} = \beta z_t + \frac{\nabla \mu_{u_t}(y_t, \theta)}{\mu_{u_t}(y_t, \theta)}, \text{ with } \beta \in (0, 1)$$

converges to a good estimate of $\nabla J(\theta)$ when β close to 1

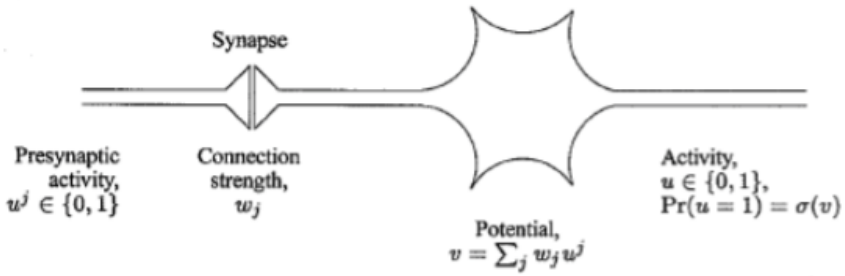
- online approach :

$$\theta_{t+1} = \theta_t + \alpha r_t z_{t+1}, \text{ with } \alpha \text{ small learning rate}$$

- still valid in a multi-agent framework :

$$\begin{aligned} z^i_{t+1} &= \beta z^i_t + \frac{\nabla \mu_{u^i_t}(y^i_t, \theta^i)}{\mu_{u^i_t}(y^i_t, \theta^i)} \\ \theta^i_{t+1} &= \theta^i_t + \alpha r_t z^i_{t+1} \end{aligned}$$

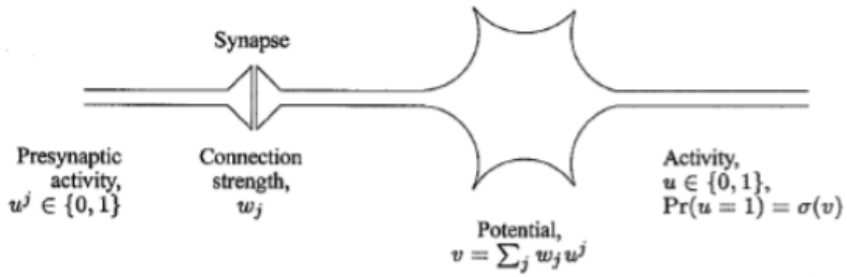
A very simple spiking neural model



Stochastic spiking neuron model :

- Actions : $u_t = 1$ fire at time t, otherwise $u_t = 0$
- Potential $v_t = \sum_j w_j u^j_{t-1}$
  - w_j is the connection strength of j^{th} synapse
  - u^j_{t-1} is the activity of j^{th} presynaptic neuron
- Firing probability $P(u_t = 1) = \sigma(v_t) = \frac{1}{1+exp(-v_t)}$

RL of optimal fires



local RL rule :

$$\begin{aligned} z_{j,t+1} &= \beta z_{j,t} + (u_t - \sigma(v_t)) u^j_{t-1} \\ w_{j,t+1} &= w_{j,t} + \alpha r_t z_{j,t+1} \end{aligned}$$

Very similar to reward modulated STDP (cf. works by Florian et al.).

With the Spike Response Model (SRM)

A more realistic neuron model with memory of past inputs

$$v_i(t) = \eta_i(t - \hat{t}_i) + \sum_j w_{ij} \sum_f \epsilon_{ij}(t - \hat{t}_i, t - t_j^f)$$

where $\hat{t}_i$ is the last spike and η_i the refractory response.
The neuron fires stochastically with

$$P(u^i(t) = 1) = \rho_i(v_i(t) - \theta_i)$$

(escape noise model), with ρ_i the firing intensity (probability density)

With the Spike Response Model (SRM)

In continuous time, we obtain from OLPOMDP (Florian 2007))

$$\begin{aligned}
\frac{dw_{ij}(t)}{dt} &= \alpha r(t) z_{ij}(t) \\
\tau_z \frac{dz_{ij}(t)}{dt} &= -z_{ij}(t) + \xi_{ij}(t) \\
\xi_{ij}(t) &= \left(\frac{\Phi_i(t)}{\rho_i(t)} - 1\right) \rho_i'(t) \sum_f \epsilon_{ij}(t - \hat{t}_i, t - t_j^f)
\end{aligned}$$

where $\Phi_i(t) = \sum_f \delta(t - t_i^f)$ is the post spike train

With the Spike Response Model (SRM)

If a spike at t_i^f follows a spike at t_j^f, Δz at t_i^f with

$$\Delta z = \frac{1}{\tau_z} \frac{\rho_i'(t_i^f)}{\rho_i(t_i^f)} \epsilon_{ij}(t_i^f - \hat{t}_i, t_i^f - t_j^f)$$

After a complete decay, with a constant r,

$$\Delta w = \alpha r \Delta z \tau_z$$

Synaptic changes are modulated by the reinforcement signal $r(t)$

modulated STDP with eligibility trace

With some additional simplifications, with an exponential learning window :

$$\begin{aligned}
\frac{dw_{ij}(t)}{dt} &= \alpha r(t) z_{ij}(t) \\
\tau_z \frac{dz_{ij}(t)}{dt} &= -z_{ij}(t) + \xi_{ij}(t) \\
\xi_{ij}(t) &= \Phi_i(t) A_+ \sum_f \exp(-\frac{t - t_j^f}{\tau_+}) + \Phi_j(t) A_- \sum_f \exp(-\frac{t - t_i^f}{\tau_-})
\end{aligned}$$

Dropping the eligibilty trace :

$$\frac{dw_{ij}(t)}{dt} = \alpha r(t) \xi_{ij}(t)$$

which has to be compared to the standard STDP model

$$\frac{dw_{ij}(t)}{dt} = \alpha \xi_{ij}(t)$$

Conclusions

- interesting perspectives for deep RL with spiking neural networks
- RL might provide some cues for better understanding biological NN
- need to integrate hierarchical network representations
- discrete event system modelling and simulation could help in analysing neural plasticity

2020 M2 internship

Modeling plastic spiking neural networks using discrete event simulators

Supervisors: Frédérick Garcia (DR INRA at MIAT), Gautier Quesnel (CR INRA at MIAT), Alexandre Muzy (DR CNRS at I3S) and Timothée Masquelier (CR CNRS at CERCO).

This internship will be a collaboration between F. Garcia, G. Quesnel, A. Muzy (all experts in discrete event simulators), and T. Masquelier (expert in computational neuroscience and spiking neural networks).

Organization: The main lab for this project is the MIAT. The candidate will also frequently visit the CERCO. We will do one online meeting per week, and one face-to-face meeting per month.

Duration: 5 months.

Starting date: early 2020.

Here we want to use the Discrete Event System Specification (DEVS) framework to model and simulate SNNs, as in ref. [1]. A DEVS model of SNNs can be specified to catch temporal mechanisms. Each DEVS neuron interacts with others according to external changes in input-output potentials (spikes) and based on their internal changes in membrane potentials. The last relative time between two changes (or computations) can be formally encoded by each neuron. The temporal activity patterns of these changes can then be modeled hierarchically and analyzed.

We will incorporate the synaptic plasticity rule known as spike-timing-dependent plasticity (STDP) in our DEVS model, which has never been done before. STDP is a now well-established physiological mechanism of activity-driven synaptic plasticity. According to STDP, synapses are reinforced (respectively depressed) when a presynaptic spike arrives before (respectively after) a postsynaptic one[2]. In other words, a neuron reinforces the connections with the afferents that contributed to triggering a postsynaptic spike, in agreement with Hebb's postulate[3].