Reinforcement Learning

Olivier Pietquin
pietquin@google.com

Google Brain

Université de Lille 1 Sciences et Technologies
CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille
IUF institut universitaire de France

Learning methods

Learning methods

Supervised Learning

- Learn a mapping between inputs and outputs;
- An oracle provides labelled examples of this mapping;

Learning methods

Supervised Learning

- Learn a mapping between inputs and outputs;
- An oracle provides labelled examples of this mapping;

Unsupervised Learning

- Learn a structure in a data set (capture the distribution);
- No oracle;

Learning methods

Supervised Learning

- Learn a mapping between inputs and outputs;
- An oracle provides labelled examples of this mapping;

Unsupervised Learning

- Learn a structure in a data set (capture the distribution);
- No oracle;

Reinforcement Learning

- Learn to Behave!
- Online Learning.
- Sequential decision making, controle.

General problem

RL is a problem (unsolved), a general paradigm, not a method !

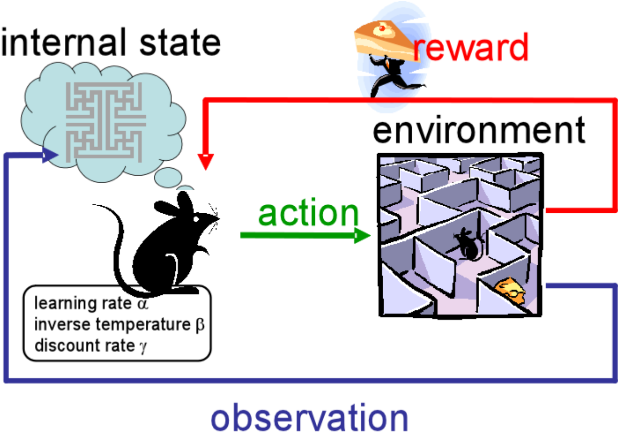


Image taken from ATR Cyber rodent project

Induced Problems

Trial-and-error learning process

- Acting is mandatory to learn.

Exploration vs Exploitation Dilemma

- Should the agent follow its current policy because it knows its consequences ?
- Should the agent explore the environment to find a better strategy ?

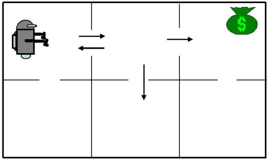
Delayed Rewards

- The results of an action can be delayed
- How to learn to sacrifice small immediate rewards to gain large long term rewards ?

Examples

Artificial problems

- Mazes or grid-worlds
- Mountain car
- Inverted Pendulum
- Games: BackGammon, Chess, Atari, Go



Real-world problems

- Man-Machine Interfaces
- Data center cooling
- Autonomous robotics

Examples I

Grid World

- State: x,y position
- Actions: up,down,right,left
- Reward: $+1$ for reaching goal state, 0 every other step

Cart Pole

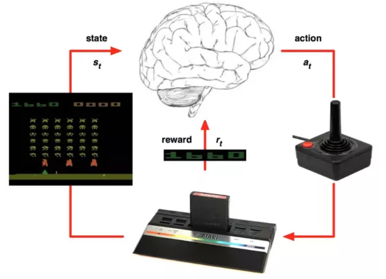
- State: angle, angular velocity
- Actions: right, left
- Reward: $+1$ for vertical position, 0 otherwise

Examples II

Chess, Go

- State: configuration of the board
- Actions: move a piece, place a stone
- Reward: +1 for winning, 0 for draw, -1 for loosing

Atari



Example: Dialogue as an MDP

The dialogue strategy is optimized at an *intention level*.

States

Dialogue states are given by the context (e.g. information retrieved, status of a database query)

Actions

Dialog acts : simple communicative acts (e.g. greeting, open question, confirmation)

Reward

User satisfaction usually estimated as a function of objective measures (e.g. dialogue duration, task completion, ASR performances)

History I

Animal Psychology - 1900

- Started with Pavlov and Skinner in the early 20th century
- 1911 : E. Thorndike, "The law of effects"

> *Of several responses made to the same situation, those which are accompanied or closely followed by satisfaction to the animal will, other things being equal, be more firmly connected with the situation, so that, when it recurs, they will be more likely to recur; those which are accompanied or closely followed by discomfort to the animal will, other things being equal, have their connections with that situation weakened, so that, when it recurs, they will be less likely to occur. The greater the satisfaction or discomfort, the greater the strengthening or weakening of the bond.*

History II

Optimal Control: 1950 - 1965

- 1957 - R. Bellman: Dynamic Programming
- 1960 - R. Howard: Dynamic Programming and Markov Processes
- 1965 - K. Astrom: Optimal control of Markov decision processes with incomplete state estimation

Machine Learning: 1980 - 1990

- 1978 - R. Sutton: Animal learning theory
- 1983 - A. Barto and R. Sutton: inverted pendulum solved with connexionist methods
- 1989 - C. Watkins: Q-learning
- 1992 - G. Tesauro: TD-Gammon

History III

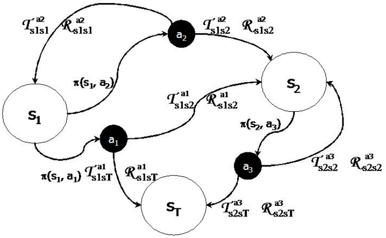
Scalling up RL: 1995 - 2005

- 1995 - Bradke & Sutton: LSTD to use linear function approximation for batch evaluation
- 2003 - Lagoudakis & Parr: LSPI same for batch control
- Long period of theoretical analysis for function approximation (Munos, Szepesvári)

Deep RL: 2005 - Today

- 2006 - M. Riedmiller: Neural Fitted-Q
- 2014 - V. Mnih: DQN learns to play Atari from pixels
- 2016 - D. Silver: AlphaGo is awarded Go 9 dans
- 2017 - D. Silver: AlphaGo learns with pure selfplay

Markov Decision Processes (MDP)



Definition (MDP)

An MDP is a Tuple $\{S, A, P_t, r_t, \gamma\}$ *such as:*

- S *is the state space;*
- A *is the action space;*
- T *is the time axis ;*
- $\mathcal{T}^a_{ss'} \in (P_t)_{t\in T}$ *is a family of markovian transition probability distributions between states conditionned on actions;*
- $(r_t)_{t\in T}$ *is a bouded familly of rewards associated to transitions*
- γ *is a discount factor*

Interpretation

At each time t of T, the agent observes the current state $s_t \in S$, performs an action $a_t \in A$ on the system wich is randomly led according to $\mathcal{T}^a_{ss'} = P_t(.|s_t, a_t)$ to a new state s_{t+1} ($P_t(s'|s, a)$ represents the probability to step into state s' after having performed action a at time t in state s), and receives a reward $r_t(s_t, a_t, s_{t+1}) \in \mathbb{R}$. with $\mathcal{R}^a_{ss'} = E[r_t|s, s', a]$

Comments

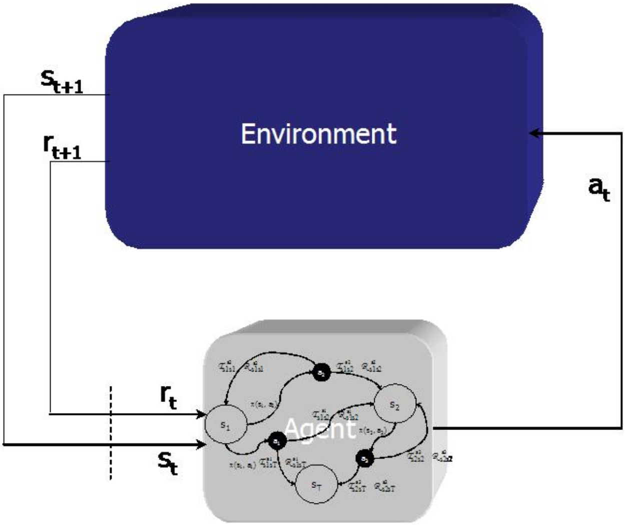
If the action selection process is deterministic

- $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t)$
- The MDP becomes a Markov chain
- A MDP is a *controled* Markov chain

MDP = environment model

- The MDP is an internal model of the world inside the learning agent
- In the perfect case, environment behaves like the MDP !
- Otherwise, modeling error occurs $\Rightarrow$ Uncertainty !

Agent's point of view



Gain : premises of local view

Definition (Cumulative reward)

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T = \sum_{i=t+1}^{T} r_i$$

Definition (Discounted cumulative reward)

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots + \gamma^{T-t+1} r_T + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

Definition (Averaged Gain)

$$R_t = \frac{1}{T-1} \sum_{i=t+1}^{T} r_i$$

Policy

$$\pi_t(a|s) : S \rightarrow \Delta^A$$

Definition (Policy or Strategy π)

The agent's policy or strategy π_t at time t is an application from S into distributions over A defining the agent's behavior (mapping between situations and actions, remember Thorndike)

Definition (Optimal Policy or Strategy π^*)

An optimal politicy or strategy π^* for a given MDP is a politicy that maximises the agent's gain

Value Function I

The agent needs a local evaluation function of the expected long term gain obtain by following a given policy

Definition (Value function for a state $V^\pi(s)$)

$$\forall s \in S \quad V^\pi(s) = E^\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s]$$

$V^\pi(s) =$ *Expected gain when starting from s and following the policy* π

Reinforcement learning

Learning this function by interaction so as to conclude on the best policy to follow.

Value Function II

The agent can also compute a local evaluation function of the expected long-term gain when choosing a given action

Definition (Action value function or Quality function $Q^\pi(s,a)$)

$$\forall s \in S, a \in A \quad Q^\pi(s,a) = E^\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a]$$

$Q^\pi(s,a) =$ *Expected gain when starting from state s, selecting action a then following policy* π

Bellman (evaluation) equation I

Note

$$V^\pi(s) = E^\pi[r(s,a) + \gamma \sum_{t=0}^{\infty} \gamma^t r(s_{t+1}, a_{t+1}) | s_0 = s, a \sim \pi(a|s)]$$

$$Q^\pi(s,a) = E^\pi[r(s,a) + \gamma \sum_{t=0}^{\infty} \gamma^t r(s_{t+1}) | s_0 = s, a_0 = a]$$

$$\forall s \in S \quad V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s|a) = Q^\pi(s, \pi(s))$$

Bellman (evaluation) equation II

Bellman evaluation equation

$$Q^\pi(s,a) = \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

$$V^\pi(s) = \sum_a \pi(s|a) \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^\pi(s')]$$

Systems of $|S|$ linear equations in $|S|$ unknowns (tabular representation).

Matrix Version

Some definitions

$$V^\pi = \begin{bmatrix} V^\pi(s^1) \\ \vdots \\ V^\pi(s^{N_s}) \end{bmatrix} \quad r_\pi = \begin{bmatrix} \sum_a \pi(s^1|a) \sum_{s'} \mathcal{T}^a_{s^1 s'} \mathcal{R}^a_{s^1 s'} \\ \vdots \\ \sum_a \pi(s^{N_s}|a) \sum_{s'} \mathcal{T}^a_{s^{N_s} s'} \mathcal{R}^a_{s^{N_s} s'} \end{bmatrix}$$

$$P_\pi = \begin{bmatrix} \sum_a \pi(s^1|a) \mathcal{T}^a_{s^1 s^1} & \cdots & \sum_a \pi(s^1|a) \mathcal{T}^a_{s^1 s^{N_s}} \\ \vdots & \ddots & \vdots \\ \sum_a \pi(s^{N_s}|a) \mathcal{T}^a_{s^{N_s} s^1} & \cdots & \sum_a \pi(s^{N_s}|a) \mathcal{T}^a_{s^{N_s} s^{N_s}} \end{bmatrix}$$

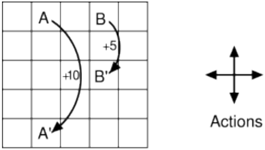
Easy version of Bellman equation

$$V^\pi = r_\pi + \gamma P_\pi V^\pi$$

Solution

$$
\begin{aligned}
V^\pi &= r_\pi + \gamma P_\pi V^\pi \\
(I - \gamma P_\pi) V^\pi &= r_\pi \\
V^\pi &= (I - \gamma P_\pi)^{-1} r_\pi
\end{aligned}
$$

Example:



(a)

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

(b)

Optimal Policy definition

Partial order

$$\pi' > \pi \quad \textit{iff} \quad \forall s \in S, \quad V^{\pi'}(s) > V^{\pi}(s)$$

Definition (Optimal Policy π^*)

An optimal policy π^* is a policy that maximizes $V^{\pi}(s)$ for every states. The associated value function is noted V^*

Formally

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

Theorem (Existence)

There exists at least one deterministic optimal policy π^* for an infinite-horizon γ-discounted MDP

Computing π^* I

Theorem (Bellman equation for $V^*(s)$)

$$
\begin{aligned}
V^*(s) &= \max_{\pi} V^{\pi}(s) = \max_{a \in A} Q^*(s,a) \\
&= \max_{a} \sum_{s'} \mathcal{T}_{ss'}^{a} [\mathcal{R}_{ss'}^{a} + \gamma V^*(s')]
\end{aligned}
$$

Theorem (Bellman Equations for $Q^*(s,a)$)

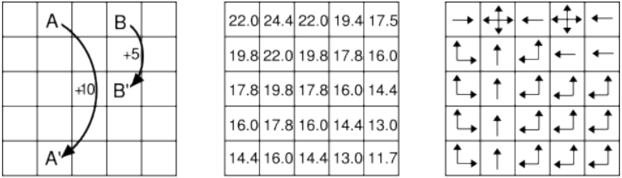
$$
\begin{aligned}
Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a) \\
&= \sum_{s'} \mathcal{T}_{ss'}^{a} [\mathcal{R}_{ss'}^{a} + \gamma V^*(s')] \\
&= \sum_{s'} \mathcal{T}_{ss'}^{a} [\mathcal{R}_{ss'}^{a} + \gamma \max_{a'} Q^*(s',a')]
\end{aligned}
$$

Computing π^* II

Given the system's dynamic (transition probabilities and immediate reward distribution), the optimal policy can be computed:

$$\forall s \in S \quad \pi^*(s) = \operatorname*{argmax}_a \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V^*(s')]$$

Example : Grid World



a) gridworld b) V^* c) π^*

Value Iteration I

Problem with the Bellman equations

System of $|S|$ non-linear equations (max) in $|S|$ unknowns

Theorem (Contraction)

The Bellman equation for $V^*(s)$ defines an operator noted B which is a contraction admitting a fixed point in $V^*(s)$

$$\begin{aligned} V_{i+1} &\leftarrow BV_i \\ V_{i+1}(s) &\leftarrow \max_a \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_i(s')] \end{aligned}$$

Note : here contraction means $\|BV_1 - BV_2\|_\ell < \gamma \|V_1 - V_2\|_\ell$ with $0 < \gamma < 1$ (example : division by 2)

Value Iteration II

Value iteration algorithm
initialize $V_0 \in \mathcal{V}$
$n \leftarrow 0$
while $\|V_{n+1} - V_n\| > \varepsilon$ **do**
for $s \in S$ **do**
$V_{n+1}(s) = \max_a \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_n(s')]$
end for
$n \leftarrow n + 1$
end while
for $s \in S$ **do**
$\pi(s) = \text{argmax}_{a \in A} \sum_{s'} \mathcal{T}^a_{ss'}[\mathcal{R}^a_{ss'} + \gamma V_n(s')]$
end for
return V_n, π

Policy Iteration I

Problem with Value Iteration

According to the value of ϵ, convergence can be very slow.

Enhancement

Even if convergence is not reached for the estimation of $V^*(s)$, the optimal policy can be found. The idea is to iterate on policies.

Definition (Greedy policy)

A policy π is greedy w.r.t. a Q-function if $\pi(s) \in \operatorname{argmax}_a Q^\pi(s, a)$

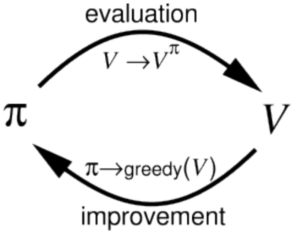
Policy Iteration II

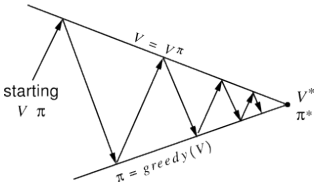
Theorem (Policy improvement theorem)

If π' is greedy w.r.t. to Q^π then $\pi' \geq \pi$

Theorem (Optimality theorem)

If π is greedy w.r.t. to its own Q-function then π is optimal.





Policy Iteration III


```
Policy iteration algorithm
```

Init $\pi_0 \in \mathcal{D}$

$n \leftarrow 0$

while $\pi_{n+1} \neq \pi_n$ **do**

  solve (Evaluation phase)

    $V_{n+1}(s) = \sum_{s'} \mathcal{T}_{ss'}^{\pi(s)}[\mathcal{R}_{ss'}^{a} + \gamma V_n(s')]$ (Linear eq.)

  for $s \in S$ **do** (Improvement phase)

    $\pi_{n+1}(s) = \text{argmax}_{a \in A} \sum_{s'} \mathcal{T}_{ss'}^{a}[\mathcal{R}_{ss'}^{a} + \gamma V_n(s')]$

  end for

  $n \leftarrow n + 1$

end while

return V_n, π_{n+1}

Asynchronous Dynamic Programming

Problem with DP

For very large state spaces, the time for one iteration can be important

Asynchronous DP

- Updating one state every k
- Introduction of *a priori* knowledge to choose the states to update.
- First premises of learning through interaction (the state selection can come from experience).

Modified policy iteration

Bellman evaluation operator is also a contraction

V^π is the fixed point of the following operator:

$$B^\pi V = r_\pi + \gamma P_\pi V$$

Evaluation can be done through this iterative process:

$$V_{i+1} \leftarrow B^\pi V_i$$

Modified Policy iteration consists in stopping evaluation after k steps and not wait for convergence. It also works with $k = 1$ (very similar to Value iteraction then).

Reinforcement Learning

Unknown environment

If the system's dynamic is not known, learning has to happen through interaction. No policy can be learnt before some information about the environment is gathered. This setting defines the Reinforcement Learning problem.

Naive Method : Adaptive DP

Learn the environment's dynamic through interaction (sampling the distributions) and apply dynamic programming.

Monte Carlo Methods

Learning $V^{\pi}(s)$ through sampling

- Random choice of a starting state $s \in S$
- Follow the policy π and observe the cumulative gain R_t
- Do this infinitly and average: $V^{\pi}(s) = E^{\pi}[R_t]$

Monte Carlo Methods

Learning $V^\pi(s)$ through sampling

- Random choice of a starting state $s \in S$
- Follow the policy π and observe the cumulative gain R_t
- Do this infinitly and average: $V^\pi(s) = E^\pi[R_t]$

Learning $Q^\pi(s, a)$ by sampling

- Random choice of a starting state $s \in S$
- Random choice of an action $a \in A$ (*exploring starts*)
- Follow policy π and observe gain R_t
- Do that infinitly and average : $Q^\pi(s, a) = E^\pi[R_t]$
- Enhance the policy : $\pi(s) = \text{argmax}_{a \in A} Q^\pi(s, a)$

Problem

Dynamic Programming

- Requires knowing the system's dynamics
- But takes the structure into account :

$$\forall s \in S \quad V^*(s) = \max_{a \in A} E(r(s,a) + \gamma \sum_{s' \in S} \mathcal{T}_{ss'}^a V^*(s'))$$

Monte Carlo

- No knowledge is necessary
- No consideration is made of the structure : $Q^\pi(s,a) = E^\pi[R_t]$
- So, the agent has to wait until the end of the interaction to improve the policy
- High variance

Temporal Differences (TD) I

TD Principle

Ideal Case (deterministic) :

$$\begin{aligned} V(s_t) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \dots \\ &= r_t + \gamma V(s_{t+1}) \end{aligned}$$

In practice :

$$\delta_t = [r_t + \gamma V(s_{t+1})] - V(s_t) \neq 0!$$

δ_t is the temporal difference error (TD error).

Note: $r(s_t, a_t) = r_t$

Note: target is now $r_t + \gamma V(s_{t+1})$ which is biased but with lower variance.

Temporal Differences (TD) II

New Evaluation method for V

Widrow-Hoff like update rule:

$$V^{t+1}(s_t) \leftarrow V^t(s_t) + \alpha \left(r_t + \gamma V^t(s_{t+1}) - V^t(s_t) \right)$$

- α is the learning rate
- $V(s_t)$ is the target

Same for Q

$$Q^{t+1}(s_t, a_t) \leftarrow Q^t(s_t, a_t) + \alpha \left(r_t + \gamma Q^t(s_{t+1}, a_{t+1}) - Q^t(s_t, a_t)\right)$$

```
SARSA
Init Q_0
for n ← 0 until N_tot − 1 do
    s_n ← StateChoice
    a_n ← ActionChoice = f(Q^{π_t}(s, a))
    Perform action a and observe s', r
    begin
        Perform action a' = f(Q^{π_t}(s', a'))
        δ_n ← r_n + γ Q_n(s'_n, a') − Q_n(s_n, a_n)
        Q_{n+1}(s_n, a_n) ← Q_n(s_n, a_n) + α_n(s_n, a_n) δ_n
        s ← s', a ← a'    end
end for
return Q_{N_tot}
```

Q-Learning

Learn π^* following π_t (off-policy)

$$Q^{t+1}(s_t, a_t) \leftarrow Q^t(s_t, a_t) + \alpha(r_t + \gamma \max_b Q^t(s_{t+1}, b) - Q^t(s_t, a_t))$$

```
Q-learning Algorithm
```

for $n \leftarrow 0$ **until** $N_{tot} - 1$ **do**

  $s_n \leftarrow$ `StateChoice`

  $a_n \leftarrow$ `ActionChoice`

  $(s'_n, r_n) \leftarrow$ `Simuler`(s_n, a_n)

  % *Update* Q_n

  begin

    $Q_{n+1} \leftarrow Q_n$

    $\delta_n \leftarrow r_n + \gamma \max_b Q_n(s'_n, b) - Q_n(s_n, a_n)$

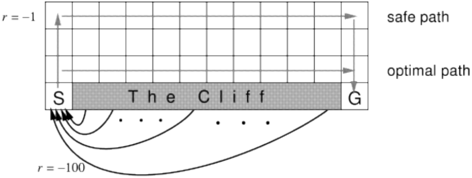
    $Q_{n+1}(s_n, a_n) \leftarrow Q_n(s_n, a_n) + \alpha_n(s_n, a_n)\delta_n$

  end

end for

return $Q_{N_{tot}}$

Q-Learning



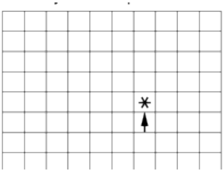
r = −1
safe path
optimal path
S
The Cliff
G
r = −100

Sarsa
Q-learning
Reward per epsiode
−25
−50
−75
−100
0
100
200
300
400
500
Episodes

Problem of TD(0) method

Problem

In case of a limited number of interactions, information propagation may not reach all the states.

Ex : grid world. 

Solution ?

Remember all interactions replay them a large number of times.

Eligibility Traces

The TD framework is based on $R_t^1 = r_{t+1} + \gamma V_t(s_{t+1})$

One can also write:

- $R_t^2 = r_t + \gamma r_{t+1} + \gamma^2 V_t(s_{t+1})$
- $R_t^n = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n V_t(s_{t+n})$

General update rule

$$\Delta V_t(s_t) = \alpha[R_t^n - V_t(s_t)]$$

Forward view I

Any average of different R_t can be used :

- $R_t^{moy} = 1/2R_t^2 + 1/2R_t^4$
- $R_t^{moy} = 1/3R_t^1 + 1/3R_t^2 + 1/3R_t^3$

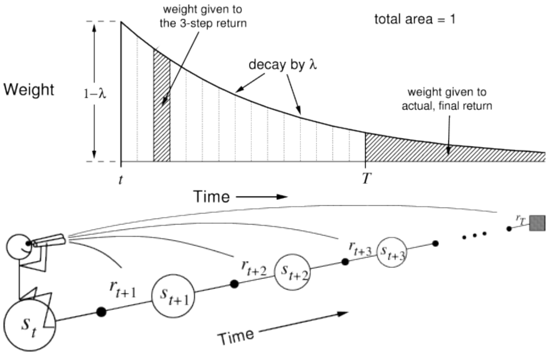
Eligibility Traces

$$R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^n$$

$$\Delta V^t(s_t) = \alpha[R_t^\lambda - V^(s_t)]$$

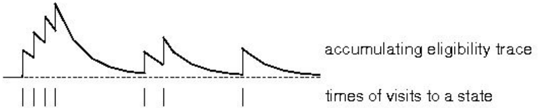
$$0 < \lambda < 1$$

Forward view II



Backward View I

A memory variable is associated to each state (state-action pair).



$$\forall s, t \; e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{si } s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{si } s = s_t \end{cases}$$

Update rule

$$\delta_t = r_t + \gamma V^t(s_{t+1}) - V^t(s_t)$$

$$\forall s \quad \Delta V^t(s) = \alpha \delta_t e_t(s)$$

Backward View II

TD(λ) et Q(λ)

- Every states are updated, the learning rate of each state being weighted by the corresponding eligibility trace;
- si $\lambda = 0$, TD(0) ;
- si $\lambda = 1$, Monte Carlo

Sarsa(λ)

$$\delta_t = r_t + \gamma Q^t(s_{t+1}, a_{t+1}) - Q^t(s_t, a_t)$$

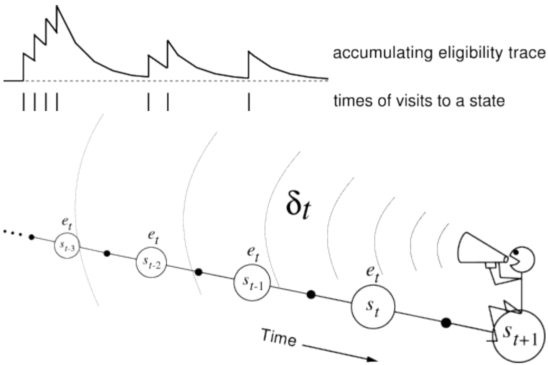
$$Q^{t+1}(s, a) = Q^t(s, a) + \alpha \delta_t e_t(s, a)$$

Watkin's $Q(\lambda)$

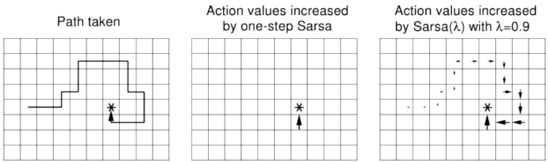
$$\delta_t = r_t + \gamma \max_b Q^t(s_{t+1}, b) - Q^t(s_t, a_t)$$

$$Q^{t+1}(s, a) = Q^t(s, a) + \alpha \delta_t e_t(s, a)$$

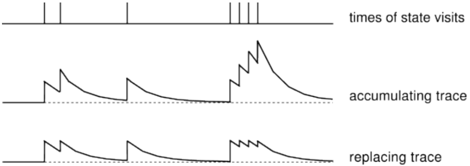
Backward View III

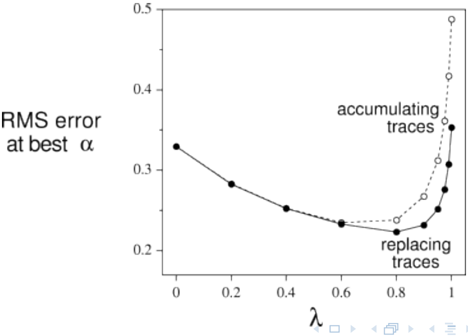


Interpretation



Replacing traces





Exploration Management

Action selection

- Greedy Selection : $a = a^* = \text{argmax}_a\, Q(s,a)$
- ϵ-greedy selection : $P(a^*) = 1 - \epsilon$
- Softmax (Gibbs or Boltzmann) $P(a) = \frac{e^{Q(a)/\tau}}{\sum_{a'} e^{Q(a')/\tau}}$

Optimistic Initialization

- Initialize the value functions with high values so as to visit unseen states thanks to action selection rules.

Uncertainty and value of information

- Take uncertainty on the values into account.
- Compute the value of information provided by exploration.

Conclusion

Good

- Optimal control without models of the physics
- Online learning

Bad

- Large state spaces
- Sample efficiency