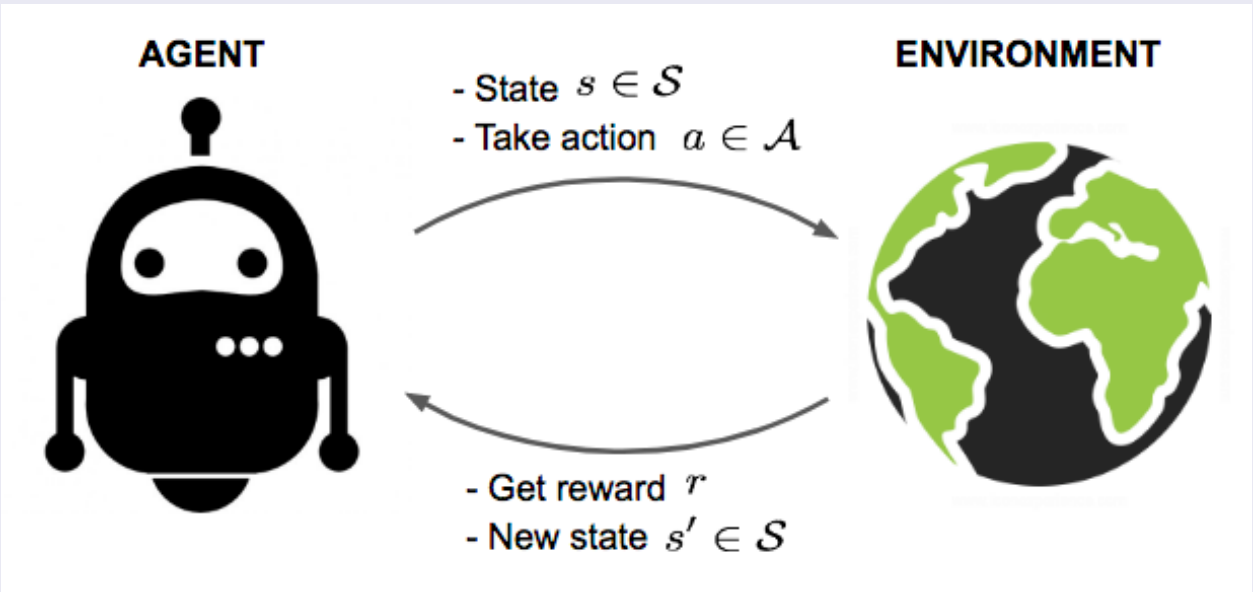
Reinforcement learning

Nicolas Wicker

Laboratoire Paul Painlevé, Université Lille, 59655, Villeneuve d'Ascq, France.

What is it about ?

Learning how to optimize one's life in a given environment



Mathematical formulation (1/3)

Markov Decision Process

- set of states S, actions A
- transition probability given two states s, s' and action a $P(s'|s, a)$
- reward probability $P(r|s, a)$ for state s and action a, reward noted $r(s, a)$

Mathematical formulation (2/3)

So what ?

The heart problem is to devise a policy π that is a strategy for an agent to guide his actions.

$$a = \pi(s)$$

a is the action returned by policy π when the current state is s. and to maximize:

$$V_\pi(s) = \mathbb{E}_{a_t \sim \pi(s_t)}\Big[\sum_{t=0}^{+\infty} \gamma^t r(s_t, a_t) | s_0 = s)\Big]$$

Mathematical formulation (3/3)

Two main cases

- the environment is know, just the policy needs to be found: planning problem
- the environment is itself unknown, the environment is explored and the policy is devised at the same time (learning problem)

Examples

Non-exhaustive list

- Backgammon (Tesauro, 1994)
- Go (Silver et al., 2007)
- packet routing (Boyan and Littmanm 1994)
- elevator control (Crites and Barto, 1996)
- helicopters control (Abbeel et al. 2010)
- video games
- targeted marketing
- fleet management (Simao et al., 2009)
- finance

Want to play too ? (1/3)

Some reinforcement learning environments make it easy and reproducible for you

- OpenAI Gym: Atari games, miscellaneous
- Gym Trading: Trading
- VIZDoom: the Doom game
- OpenSpiel (Deepmind): board games (Backgammon, chess, go)
- ns3-gym: computer networks
- RecoGym: e-commerce advertising
- OpenSim-RL: Biomechanics
- Textworld: text based games
- pysc2: Starcraft video game

Want to play too ? (2/3)

We focus on OpenAI Gym

```
import gym
env = gym.make('FrozenLake-v0')

print(env.observation_space)
state=env.reset()
print(state)
for _ in range(10):
    env.render()
    input("##################################################")
    next_state, reward, endOfEpisode, info= \
    env.step(env.action_space.sample()) # take a random action

    print(next_state)
env.close()
```

Want to play too ? (3/3)

We focus on OpenAI Gym

```
import gym
env = gym.make('FrozenLake-v0')
env.reset()
endOfEpisode = False
while not endOfEpisode:
    env.render()
    input("##############################################")
    next_state, reward, endOfEpisode, info = \
    env.step(env.action_space.sample())
    print(reward)

env.close()
```

Bellman's equations

Two core equations

- $V^*(s) = \max_{a \in A} \left\{ \mathbb{E}\big[r(s,a)\big] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right\}$
- $V_\pi(s) = \mathbb{E}_{a \sim \pi(s)}[r(s,a)] + \gamma \sum_{s'} P(s'|s,\pi(s)) V_\pi(s')$

where π is any policy and π^* the optimal one.

One useful notation

$$Q_\pi(s,a) = \mathbb{E}[r(s,a)] + \gamma \sum_{s'} P(s'|s,\pi(s)) V_\pi(s')$$

Policy evaluation (1/3)

Proposition

The policy value is given by: $(I - \gamma P)^{-1} R$

Proof.

The Bellman equation can be rewritten as: $(I - \gamma P)V = R$.
Computing the infinity norm gives:

$$\begin{aligned} \|P\|_\infty &= \max_{x \neq 0} \frac{\|Px\|_\infty}{\|x\|_\infty} \\ &= \max_s \sum_{s'} |P_{ss'}| \\ &= \max_s \sum_{s'} P(s'|s, \pi(s)) = 1 \end{aligned}$$

Thus $\|\gamma P\|_\infty = \gamma < 1$, the eigenvalues of γP are then less than 1 and $I - \gamma P$ is thus invertible. □

Policy evaluation (2/3)

Remark

This may cost a lot of computation: $O(n^3)$ so an alternative may be preferred following an iterative algorithm.

Policy evaluation (3/3)

By iteration

$\forall s, V(s) \leftarrow \text{random}$ and $V(\text{terminal}) \leftarrow 0$
repeat
  $\Delta \leftarrow 0$
  for $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max\big(\Delta, |v - V(s)|\big)$
  end for
until $\Delta < \theta$

Value iteration (1/4)

Algorithm

$V \leftarrow V_0$
while $\|V - \Phi(V)\| \geq \frac{(1-\gamma)\epsilon}{\gamma}$ **do**
  $V \leftarrow \Phi(V)$
end while

with $\Phi(V) = \max_\pi \{R_\pi + \gamma P_\pi V\}$, $(P_\pi)_{ss'} = P(s'|s, \pi(s))$ and $(R_\pi)_s = \mathbb{E}\big[r(s, \pi(s))\big]$

Value iteration (2/4)

Proposition

The sequence defined by $V_{n+1} = \Phi(V_n)$ *converges to* V^*.

Proof.

Pick two values function $U, V \in \mathbb{R}^{|S|}$, then:

$$\begin{aligned}
\Phi(V)(s) - \Phi(U)(s) \leq & \Phi(V)(s) - \Big(\mathbb{E}\big[r(s, a^*(s)) + \\
& \sum_{s' \in S} P(s'|s, a^*(s)) U(s')\big]\Big) \text{ where } a^*(s) \\
& \text{is the maximizing action in the definition} \\
& \text{of } \Phi(V)(s) \\
\leq & \gamma \sum_{s' \in S} P(s'|s, a^*(s))\big[V(s') - U(s')\big] \\
\leq & \gamma \sum_{s' \in S} P(s'|s, a^*(s)) \|V - U\|_\infty = \gamma \|V - U\|_\infty
\end{aligned}$$

Value iteration (3/4)

Proof.

Now, noting that V^* is a fixed point for Φ and that Φ is $\gamma-$ contracting concludes the proof as $\mathbb{R}^{|S|}$ is a complete space for $\|.\|_\infty$. $\square$

Value iteration (4/4)

Proposition

The resulting policy of the algorithm is ϵ-optimal.

Proof.

$$\begin{aligned}\|V^* - V_{n+1}\|_\infty &\leq \|V^* - \Phi(V_{n+1})\|_\infty + \|\Phi(V_{n+1}) - V_{n+1}\|_\infty \\ &= \|\Phi(V^*) - \Phi(V_{n+1})\|_\infty + \|\Phi(V_{n+1}) - \Phi(V_n)\|_\infty \\ &\leq \gamma\|V^* - V_{n+1}\|_\infty + \gamma\|V_{n+1} - V_n\|_\infty\end{aligned}$$

Then,

$$\|V^* - V_{n+1}\|_\infty \leq \frac{\gamma}{1-\gamma}\|V_{n+1} - V_n\| \leq \epsilon$$

using the condition ending the value iteration algorithm. □

Policy improvement (1/2)

Proposition

$q_\pi(s, \pi'(s)) \geq \nu_\pi$ *implies* $\nu_{\pi'}(s) \geq \nu_\pi(s)$

Proof.

$$
\begin{aligned}
\nu_\pi(s) \leq & q_\pi(s, \pi'(s)) \\
= & \mathbb{E}\Big[R_{t+1} + \gamma \nu_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\Big] \\
= & \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma \nu_\pi(S_{t+1}) | S_t = s\Big] \\
\leq & \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s\Big] \text{ by hypothesis} \\
= & \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma \mathbb{E}_{\pi'}\Big[R_{t+2} + \gamma \nu_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1}\Big] \\
& | S_t = s\Big] \\
= & \mathbb{E}_{\pi'}\Big[R_{t+1} + \gamma R_{t+2} + \gamma^2 \nu_\pi(S_{t+2}) | S_t = s\Big]
\end{aligned}
$$

Policy improvement (2/2)

Proof.

By iterating, we get:

$$\begin{aligned}
\nu_\pi(s) \leq & \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 \nu_\pi(S_{t+3}) | S_t = s\right] \\
& \vdots \\
\leq & \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} \ldots | S_t = s\right] \\
= & \nu_{\pi'}(s)
\end{aligned}$$

□

Policy iteration (1/3)

Algorithm

$\pi \leftarrow \pi_0$
$\pi' \leftarrow NULL$
while $\pi \neq \pi'$ **do**
  $V \leftarrow V_\pi$; policy evaluation;
  $\pi' \leftarrow \pi$
  $\pi \leftarrow \arg\max_\pi \{R_\pi + \gamma P_\pi V\}$
end while

Policy iteration (2/3)

Proposition

If $(V_n)_n$ is a sequence of policy values, then $V_n \leq V_{n+1} \leq V^*$.

Proof.

Let π_{n+1} be the policy improvement after the n^{th} step. As remarked earlier $\|\gamma P\|_\infty = \gamma < 1$ so:

$$\left(I - \gamma P_{\pi_{n+1}}\right)^{-1} = \sum_{k=0}^{+\infty} \left(\gamma P_{\pi_{n+1}}\right)^k$$

Thus,

$$\left(I - \gamma P_{\pi_{n+1}}\right)^{-1} (Y - X) = \sum_{k=0}^{+\infty} \left(\gamma P_{\pi_{n+1}}\right)^k (Y - X) \geq 0$$

if $Y \geq X$, meaning that the iteration preserves the ordering of policy evaluations.

Policy iteration (3/3)

Proof.

By definition of π_{n+1}, we have:

$$R_{\pi_{n+1}} + \gamma P_{\pi_{n+1}} V_n \geq R_{\pi_n} + \gamma P_{\pi_n} V_n = V_n$$

Thus,

$$R_{\pi_{n+1}} \geq \left(I - \gamma P_{\pi_{n+1}}\right) V_n$$

and by order preservation, we have:

$$V_{n+1} = \left(I - \gamma P_{\pi_{n+1}}\right)^{-1} R_{\pi_{n+1}} \geq V_n$$

Monte-Carlo policy first-time visit evaluation (1/2)

Idea

- It's simple, just repeat simulation with a given policy and assess for each state what's the average outcome it gets after it has been encountered once.
- some variants exist (every-visit) where all visits to a state are considered

Monte-Carlo policy first-time visit evaluation (2/2)

$\forall s \in \mathcal{S}, V(s) \in \mathbb{R}$ arbitrarily
$\text{Returns}(s) \leftarrow$ an empty list, $\forall s \in \mathcal{S}$
for each episode **do**
  generate an episode following $\pi : S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t does not appear in $S_0, S_1, \ldots, S_{t-1}$ **then**
      append G to $\text{Returns}(S_t)$
      $V(S_t) \leftarrow \text{average}(\text{Returns}(S_t))$
    end if
  end for
end for

Monte-Carlo ES (exploring starts) (1/3)

Intuition

- similar to policy evaluation
- pick the pair state,action
- choose the action maximizing the Q function

Monte-Carlo ES (exploring starts) (2/3)

$\forall s \in \mathcal{S}, V(s) \in \mathbb{R}$ arbitrarily
$\text{Returns}(s, a) \leftarrow$ an empty list, $\forall s \in \mathcal{S}$ and $a \in \mathcal{A}$
for each episode **do**
  generate an episode following $\pi : S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t does not appear in $S_0, S_1, \ldots, S_{t-1}$ **then**
      append G to $\text{Returns}(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$
      $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$
    end if
  end for
end for

Monte-Carlo ES (exploring starts) (3/3)

Funny fact

Convergence of Monte-Carlo ES is not proven.

However, some partial results are available as for instance:

- for $\gamma < 1$ and uniform initialization of state/action (Tsitsiklis, 2002)
- no cycle in optimal policy (Wang and Ross, arxiv 2020)

Monte-Carlo on-policy control (1/3)

Idea

- assumption on the initial starts is not very realistic
- we have to find a way of exploring all states without this assumption: softening the choice of the action, making the policy ϵ-greedy

Monte-Carlo on-policy control (2/3)

$\forall s \in \mathcal{S}, V(s) \in \mathbb{R}$ arbitrarily
$\text{Returns}(s,a) \leftarrow$ an empty list, $\forall s \in \mathcal{S}$ and $a \in \mathcal{A}$
for each episode **do**
  generate an episode following $\pi : S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  for each step $t = T-1, T-2, \ldots, 0$ **do**
    $G \leftarrow \gamma G + R_{t+1}$
    if S_t does not appear in $S_0, S_1, \ldots, S_{t-1}$ **then**
      append G to $\text{Returns}(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(\text{Returns}(S_t, A_t))$
      $A^* \leftarrow \arg\max_a Q(S_t, a)$
      for $a \in \mathcal{A}(S_t)$ **do**

$$\pi(A|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

      end for
    end if
  end for
end for

Monte-Carlo on-policy control (3/3)

This modification improves the policy

$$
\begin{aligned}
q_\pi(s, \pi'(s)) &= \sum_a \pi'(a|s) q_\pi(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \max_a q_\pi(s,a) \\
&\geq \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + (1-\epsilon) \sum_a \frac{\pi(a|s) - \frac{\epsilon}{|\mathcal{A}(s)|}}{1-\epsilon} q_\pi(s,a) \\
&= \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) - \frac{\epsilon}{|\mathcal{A}(s)|} \sum_a q_\pi(s,a) + \\
&\quad \sum_a \pi(a|s) q_\pi(s,a) \\
&= \nu_\pi(s)
\end{aligned}
$$

implying that the policy is not worsened as $\nu_{\pi'}(s) \geq \nu_\pi(s)$

Application of Monte-Carlo ES to Blackjack (1/2)

Blackjack

- game between a player and a dealer (bank)
- goal : be the closest to 21, face cards value is 10, ace is 10 or 1 according to the situation, other cards have their nominal value
- the player can choose to stick (he stops) or hit (he asks for an additional card) and he sees the first card of the dealer
- the decision of the player depends on three parameters:
  - dealer sum $1, \ldots, 10$
  - its sum
  - having a usable ace or not

Application of Monte-Carlo ES to Blackjack (2/2)

Results

- 200 states
- $\gamma = 1$ (no discount)
- outcome after each episode: $\{-1, 0, 1\}$
- π^* is well approximated after 500000 games

Temporal-difference learning

Different from Monte-Carlo, here we bootstrap

- in Monte-Carlo we have to wait for the final outcome, it works but costs time
- here, we modify (improve) the policy-evaluation (or/and control) whenever we get information

TD(0) for evaluation (1/4)

A simple update rule

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

whereas for Monte-Carlo we would have:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[G_t - V(S_t)\Big]$$

TD(0) for evaluation (2/4)

Initialize $V(s), \forall s \in S$
for each episode **do**
  initialize S
  for each step of epoch t **do**
    $A \leftarrow$ action given by π for S
    take action A, observe R, S'
    $V(S) \leftarrow V(S) + \alpha\left[r' + \gamma V(S') - V(S)\right]$
    $S \leftarrow S'$
  end for
end for

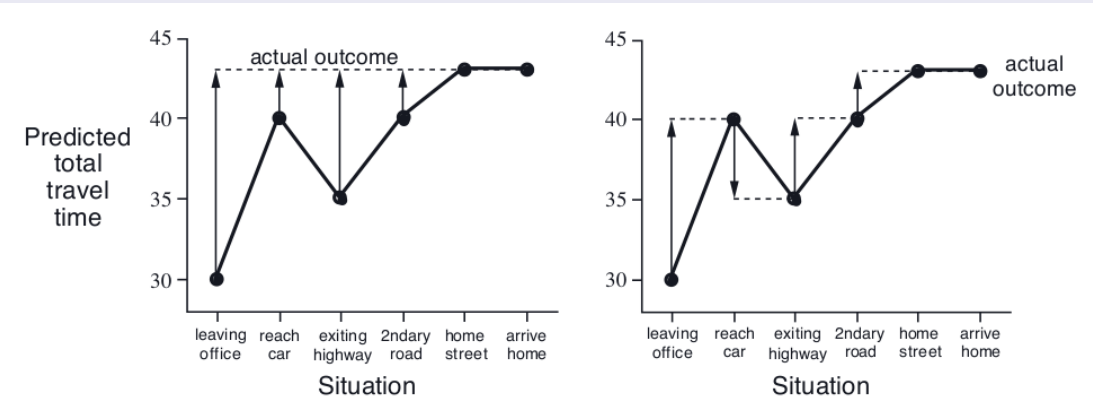
TD(0) for evaluation (3/4)

Difference with a day-life example: driving home

State	Elapsed Time (minutes)	Predicted Time to go	Predicted Total Time
leaving office	0	30	30
reach car, raining	5	35	40
exiting highway	20	15	35
2ndary road, behind truck	30	10	40
entering home street	40	3	43
arrive home	43	0	43

TD(0) for evaluation (4/4)

Difference with a day-life example: driving home



SARSA (TD(0) online policy)

---

for each episode **do**
  initialize S
  choose A from A using the policy derived from Q (e.g. ϵ-greedy)
  for each step of episode **do**
    take action A, observe R, S'
    choose A' from S' using policy derived from Q
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S',A') - Q(S,A)\big]$
    $S \leftarrow S', A \leftarrow A'$
  end for
end for

Q-learning (TD(0) off-policy) (1/2)

initialize $Q(s, a) \forall s \in S, a \in A(s)$
for each episode **do**
  initialize S
  for each step of epoch t **do**
    choose A from S using the policy derived from Q (e.g. ϵ-greedy)
    take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\left[R + \gamma \max_a Q(S', a) - Q(S, A)\right]$
    $S \leftarrow S'$
  end for
end for

What difference between online and off-policy ?

Definitions

- The behaviour policy is the one enabling the learning.
- The target policy is the one you want to adopt once the learning is finished.

Short answer

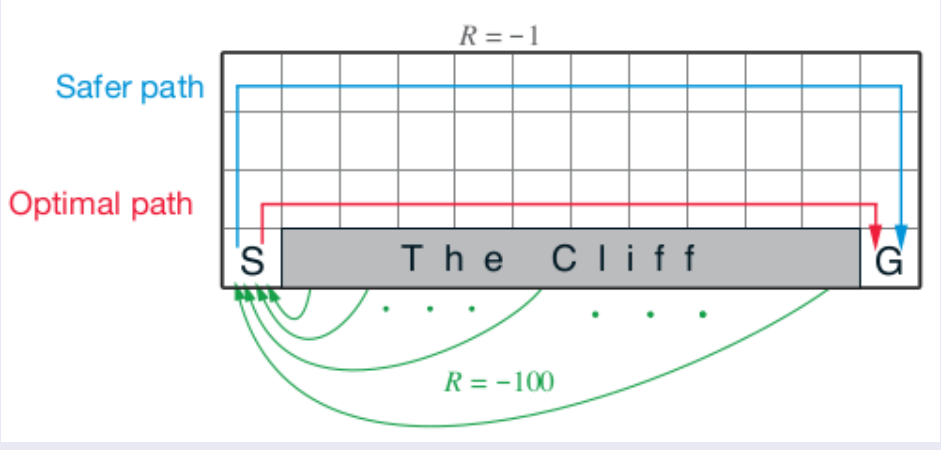
- Online policy: behaviour policy=target policy
- Off-policy: behaviour policy≠target policy

Supplementary answers

- on-policy converges to a near-optimal policy (unless tweaking like $\epsilon \to 0$)
- on-policy is fine if you prefer safe exploration (c.f. cliff problem)

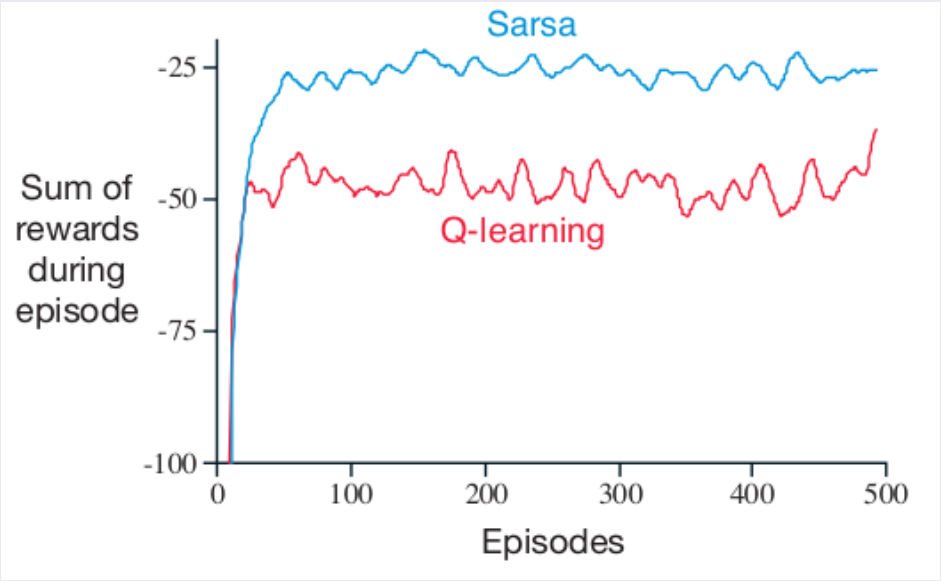
Off policy problem

The cliff (1/2)



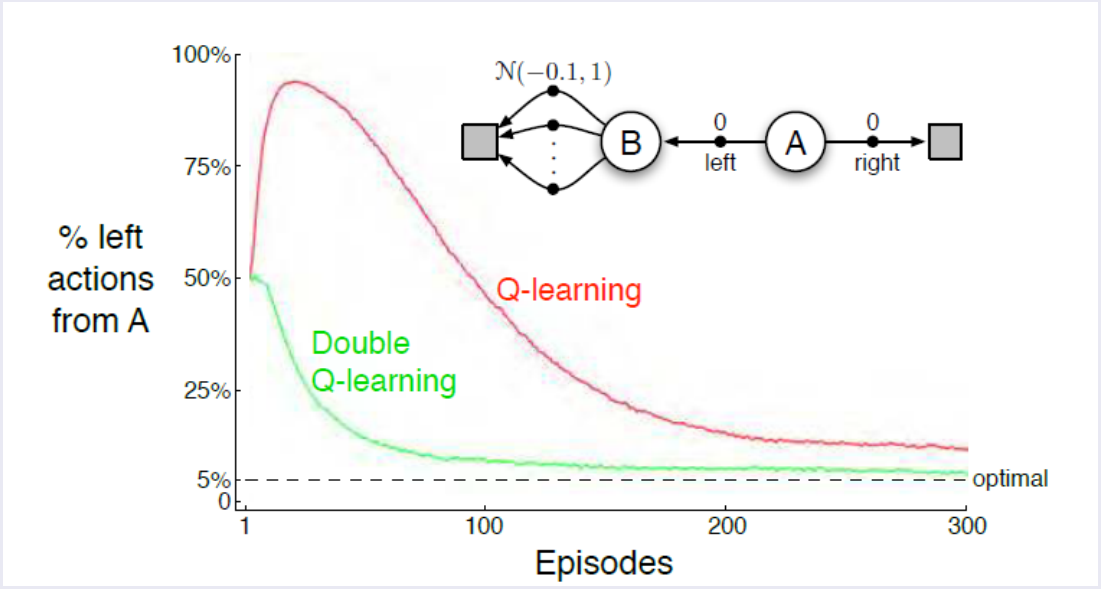
Off policy problem

The cliff (2/2)



Double Q-learning (1/3)

How you can be mistaken by a single experience.



In this setting, the agent always starts in A, the optimal strategy is to go right.
However, if once the agent gets a good reward on the left, he will keep on going to the left for a long time.

Double Q-learning (2/3)

The answer: double Q-learning

The action evaluation of a first agent is provided by a second agent, that way a bad action is more difficult to be taken just at random. The update rule is:

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big[R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big]$$

with probability 0.5, in the other case roles of agents 1 and 2 are switched.

Double Q-learning (3/3)

initialize $Q_1(s,a)$ and $Q_2(s,a) \forall s \in S^+, a \in A(s)$
for each episode **do**
  initialize S
  for each step of epoch t **do**
    choose A from S using the policy derived from $Q_1 + Q_2$
    take action A, observe R, S'
    if $B(0.5) == 1$ **then**

$$Q_1(S,A) \leftarrow Q_1(S,A) + \alpha\Big[R + \gamma Q_2\big(S', \arg\max_a Q_1(S',a)\big) - Q_1(S,A)\Big]$$

    else

$$Q_2(S,A) \leftarrow Q_2(S,A) + \alpha\Big[R + \gamma Q_1\big(S', \arg\max_a Q_2(S',a)\big) - Q_2(S,A)\Big]$$

    end if
  end for
end for

Solving the Frozen-Lake problem (1/4)

```
epsilon=0.1;gamma=0.9;alpha=0.01
Q = numpy.zeros([env.observation_space.n, env.action_space.n])

nbEpisodes=100000
for i in range(0, nbEpisodes):
    state=env.reset()

    firstState=state
    endOfEpisode = False
    nbSteps=0
    while not endOfEpisode:
        if random.uniform(0, 1) < epsilon:
            action = env.action_space.sample()
        else:
            action = numpy.argmax(Q[state])

        next_state, reward, endOfEpisode, info = env.step(action)

        next_max = numpy.max(Q[next_state])

        Q[state,action] = (1 - alpha) * Q[state,action] + alpha * (reward + gamma * next_max)

        state = next_state
        nbSteps=nbSteps+1
```

Solving the Frozen-Lake problem (2/4)

```
#evaluation
averageNumberSuccesses=0
for i in range(0, nbEpisodes):
    state=env.reset()

    endOfEpisode = False
    while not endOfEpisode:
        action = numpy.argmax(Q[state])

        next_state, reward, endOfEpisode, info = env.step(action)

        state = next_state

    if reward==1:
        averageNumberSuccesses=averageNumberSuccesses+1
averageNumberSuccesses=averageNumberSuccesses/nbEpisodes

print(averageNumberSuccesses)
```

Solving the Frozen-Lake problem (3/4)

```
#evaluation random strategy
averageNumberSuccesses=0
for i in range(0, nbEpisodes):
    state=env.reset()

    endOfEpisode = False
    while not endOfEpisode:
        next_state, reward, endOfEpisode, info = env.step(env.action_space.sample())

        state = next_state

    if reward==1:
        averageNumberSuccesses=averageNumberSuccesses+1
averageNumberSuccesses=averageNumberSuccesses/nbEpisodes

print(averageNumberSuccesses)
```

Solving the Frozen-Lake problem (4/4)

Results: percentage of times the goal is found

- Qlearning: 73.1
- Randomly: 1.4