Reinforcement Learning

Lecture 9 : Bandit tools for Reinforcement Learning

Rémy Degenne
(remy.degenne@inria.fr)

Centrale Lille, 2022/2023

From bandit to RL

Solve a multi-armed bandit problem
$=$ maximize rewards in a MDP with one state

The bandit world

- several principles for exploration/exploitation
- efficient algorithms (UCB, Thompson Sampling)
- with regret guarantees

RL algorithms so far

- ϵ-greedy exploration
- algorithms with (sometimes) convergence guarantees that are not very efficient

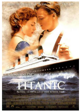
vs. (more) efficient algorithms with little theoretical understanding

Question : can we be inspired by bandit algorithms to

- propose new RL algorithms
- ... with theoretical guarantees ?

Outline

A more general bandit problem



In each time step t :

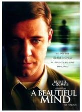
- a *context* $x_t \in \mathcal{X}$ is observed
  (e.g. the history of user t, characteristics of the movies)
- an arm $a_t \in \mathcal{A}_t$ is chosen by the algorithm
  (e.g. a movie in the catalog which is currently available)
- a reward $r_t = f(x_t, a_t) + \varepsilon_t$ is observed

Observations :

→ the mean rewards depends on the chosen arm AND on the context

→ the context plays the role of a *state*
(however the next state does not necessarily depend on our actions)

A more general bandit problem



user t : characteristic vector $u_t \in \mathbb{R}^p$

movie a : characteristic vector $x_a \in \mathbb{R}^{p'}$

➜ build a user-movie feature vector $x_{a,t} \in \mathbb{R}^d$

In each time step :

- the agent chooses an "arm" $x_t \in \mathcal{X}_t = \{(x_{a,t})_{a \in \mathcal{A}_t}\} \subseteq \mathbb{R}^d$
- and gets a reward $r_t = f(x_t) + \varepsilon_t$

where $f : \mathbb{R}^d \to \mathbb{R}$ is a regression function and $\mathbb{E}[\varepsilon_t | \mathcal{F}_{t-1}] = 0$.

Contextual linear bandits

In each round t, the agent

- receives a (finite) set of arms $\mathcal{X}_t \subseteq \mathbb{R}^d$
- chooses an arm $x_t \in \mathcal{X}_t$
- gets a reward $r_t = \theta_\star^\top x_t + \varepsilon_t$

where

- $\theta_\star \in \mathbb{R}^d$ is an unknown regression vector
- ε_t is a centered noise, independent from past data

Assumption : σ^2- sub-Gaussian noise

$$\forall \lambda \in \mathbb{R}, \ \mathbb{E}\left[e^{\lambda X}\right] \leq e^{\frac{\lambda^2 \sigma^2}{2}}$$

e.g., Gaussian noise, bounded noise.

Contextual linear bandits

In each round t, the agent

- receives a (finite) set of arms $\mathcal{X}_t \subseteq \mathbb{R}^d$
- chooses an arm $x_t \in \mathcal{X}_t$
- gets a reward $r_t = \theta_\star^\top x_t + \varepsilon_t$

where

- $\theta_\star \in \mathbb{R}^d$ is an unknown regression vector
- ε_t is a centered noise, independent from past data

(Pseudo)-regret for contextual bandit

maximizing expected total reward $\leftrightarrow$ minimizing the expectation of

$$\mathcal{R}_T = \sum_{t=1}^{T} \left(\max_{x \in \mathcal{X}_t} \theta_\star^\top x - \theta_\star^\top x_t \right)$$

→ in each round, comparison to a possibly different optimal action !

Tools for solving linear bandits

Algorithms will rely on estimates / confidence regions / posterior distributions for $\theta_\star \in \mathbb{R}^d$.

- design matrix (with regularization parameter $\lambda > 0$)

$$B_t^\lambda = \lambda I_d + \sum_{s=1}^{t} x_s x_s^\top$$

- regularized least-square estimate

$$\hat{\theta}_t^\lambda = \left(B_t^\lambda\right)^{-1} \left(\sum_{s=1}^{t} r_t x_t \right)$$

- estimate of the expected reward of an arm $x \in \mathbb{R}^d$: $x^\top \hat{\theta}_t^\lambda$

➜ sufficient for Follow the Leader, but not for smarter algorithms !

A Bayesian view on Linear Regression

Bayesian model :

- likelihood : $r_t = \theta_\star^\top x_t + \varepsilon_t$
- prior : $\theta_\star \sim \mathcal{N}(0, \kappa^2 \mathrm{I}_d)$

Assuming further that the noise is Gaussian : $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$, the posterior distribution of $\theta_\star$ has a closed form :

$$\theta_\star | x_1, r_1, \dots, x_t, r_t \ \sim \ \mathcal{N}\left(\hat{\theta}_t^\lambda, \sigma^2 \left(B_t^\lambda\right)^{-1}\right)$$

with

- $B_t^\lambda = \lambda \mathrm{I}_d + \sum_{s=1}^t x_s x_s^\top$
- $\hat{\theta}_t^\lambda = \left(B_t^\lambda\right)^{-1} \left(\sum_{s=1}^t r_s x_s\right)$ is the regularized least square estimate

with a regularization parameter $\lambda = \frac{\sigma^2}{\kappa^2}$.

Thompson Sampling for Linear Bandits

Recall the Thompson Sampling principle :

> "draw a possible model from the posterior distribution and act optimally in this sampled model"

Thompson Sampling in linear bandits

In each round $t+1$,

$$
\begin{aligned}
\tilde{\theta}_t &\sim \mathcal{N}\left(\hat{\theta}_t^\lambda, \sigma^2 \left(B_t^\lambda\right)^{-1}\right) \\
x_{t+1} &= \underset{x \in \mathcal{X}_{t+1}}{\operatorname{argmax}}\ x^\top \tilde{\theta}_t
\end{aligned}
$$

Numerical complexity : one needs to draw a sample from a multivariate Gaussian distribution, e.g.

$$\tilde{\theta}_t = \hat{\theta}_t^\lambda + \sigma \left(B_t^\lambda\right)^{-1/2} X$$

where X is a vector with d independent $\mathcal{N}(0,1)$ entries.

Thompson Sampling for Linear Bandits

Recall the Thompson Sampling principle :

> "draw a possible model from the posterior distribution and act optimally in this sampled model"

Thompson Sampling in linear bandits

In each round $t+1$,

$$\begin{aligned} \tilde{\theta}_t &\sim \mathcal{N}\left(\hat{\theta}_t^\lambda, \sigma^2 \left(B_t^\lambda\right)^{-1}\right) \\ x_{t+1} &= \underset{x \in \mathcal{X}_{t+1}}{\operatorname{argmax}} \; x^\top \tilde{\theta}_t \end{aligned}$$

Regret guarantees : [Agrawal and Goyal, 2013] prove that (a variant of) Thompson Sampling attains sub-linear regret :

$$\mathcal{R}_T(\mathrm{TS}) = \mathcal{O}\left(d^{3/2}\sqrt{T}\right) \quad \text{with high probability}$$

Outline

Regret minimization

For simplicity, we will define regret for **episodic MDPs**, in which

$$V^\pi(s) = V_1^\pi(s) = \mathbb{E}^\pi\left[\sum_{h=1}^{H} r(s_t, a_t) \middle| s_1 = s\right].$$

For each episode $t \in \{1, \dots, T\}$, an episodic RL algorithm

- starts in some initial state $s_1^t \sim \rho$
- selects a policy π^t (based on observations from past episodes)
- uses this policy to generate an episode of length H :

$$s_1^t, a_1^t, r_1^t, s_2^t, \dots, s_H^t, a_H^t, r_H^t$$

where $a_h^t = \pi_h^t(s_h^t)$ and $(r_h^t, s_{h+1}^t) = \mathrm{step}(s_h^t, a_h^t)$

Definition

The (pseudo)-regret of an episodic RL algorithm $\pi = (\pi^t)_{t\in\mathbb{N}}$ in T episodes is

$$\mathcal{R}_T(\pi) = \sum_{t=1}^{T}\left[V^\star(s_1^t) - V^{\pi^t}(s_1^t)\right].$$

Regret minimization

For simplicity, we will define regret for **episodic MDPs**, in which

$$V^\pi(s) = V_1^\pi(s) = \mathbb{E}^\pi\left[\sum_{h=1}^{H} r(s_t, a_t) \middle| s_1 = s\right].$$

For each episode $t \in \{1, \ldots, T\}$, an episodic RL algorithm

- starts in some initial state $s_1^t \sim \rho$
- selects a policy π^t (based on observations from past episodes)
- uses this policy to generate an episode of length H :

$$s_1^t, a_1^t, r_1^t, s_2^t, \ldots, s_H^t, a_H^t, r_H^t$$

where $a_h^t = \pi_h^t(s_h^t)$ and $(r_h^t, s_{h+1}^t) = \text{step}(s_h^t, a_h^t)$

Definition

The (pseudo)-regret of an episodic RL algorithm $\pi = (\pi^t)_{t\in\mathbb{N}}$ in T episodes is

$$\mathcal{R}_T(\pi) = \sum_{t=1}^{T}\left[\max_a r(s_1, a) - r(s_1, a_1^t)\right] \quad H = 1, \text{single state } s_1.$$

Regret minimization

For simplicity, we will define regret for **episodic MDPs**, in which

$$V^\pi(s) = V_1^\pi(s) = \mathbb{E}^\pi\left[\sum_{h=1}^{H} r(s_t, a_t) \,\middle|\, s_1 = s\right].$$

For each episode $t \in \{1, \dots, T\}$, an episodic RL algorithm

- starts in some initial state $s_1^t \sim \rho$
- selects a policy π^t (based on observations from past episodes)
- uses this policy to generate an episode of length H :

$$s_1^t, a_1^t, r_1^t, s_2^t, \dots, s_H^t, a_H^t, r_H^t$$

where $a_h^t = \pi_h^t(s_h^t)$ and $(r_h^t, s_{h+1}^t) = \mathrm{step}(s_h^t, a_h^t)$

Definition

The (pseudo)-regret of an episodic RL algorithm $\pi = (\pi^t)_{t\in\mathbb{N}}$ in T episodes is

$$\mathcal{R}_T(\pi) = \sum_{t=1}^{T}\left[\mu^\star - \mu_{a_1^t}\right] \quad H = 1, \text{single state } s_1.$$

Reminder : Minimizing regret in bandits

Small regret requires to introduce the right amount of exploration, which can be done with

- ϵ-greedy

explore uniformly with probability ϵ, otherwise trust the estimated model

- Upper Confidence Bounds algorithms

act as if the optimistic model were the true model

- Thompson Sampling

act as if a model sampled from the posterior distribution were the true model

What is wrong with ε-greedy in RL ?

Example : Q-Learning with ε-greedy

- ➜ ε-greedy exploration

$$a_t = \begin{cases} \operatorname{argmax}_{a \in \mathcal{A}} \hat{Q}_t(s_t, a) & \text{with probability } 1 - \varepsilon_t \\ \sim \mathcal{U}(\mathcal{A}) & \text{with probability } \epsilon_t \end{cases}$$

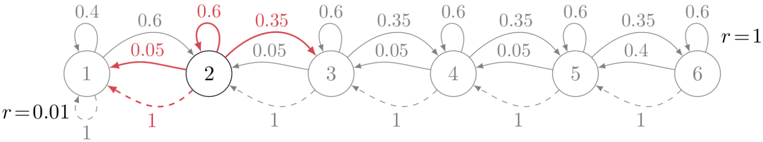
- ➜ Q-Learning update

$$\hat{Q}_t(s_t, a_t) = \hat{Q}_{t-1}(s_t, a_t) + \alpha_t \left(r_t + \gamma \max_b \hat{Q}_{t-1}(s_t, b) - \hat{Q}_{t-1}(s_t, a_t) \right)$$

⚠ $\hat{Q}_t(s, a)$ is *not* an unbiased estimate of $Q^\star(s, a)$...
(except in the bandit case)

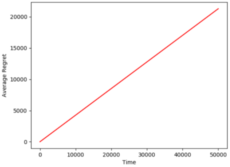
What is wrong with ε-greedy ?

The RiverSwim MDP :

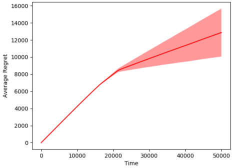


⚠ ε can be hard to tune...

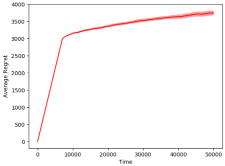
What is wrong with ε-greedy ?



$$\epsilon_t = 0.5$$



$$\epsilon_t = \frac{\epsilon_0}{(N(s_t) - 1000)^{2/3}}$$



$$\epsilon_t = \begin{cases} 1 & \text{if } t < 7000 \\ \frac{\epsilon_0}{\sqrt{N(s_t)}} & \text{otherwise} \end{cases}$$

credit : Alessandro Lazaric

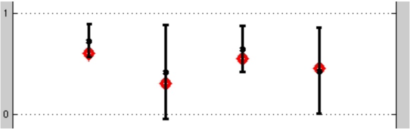
⚠ ε-greedy performs undirected exploration

- alternative : model-based methods in which exploration is targeted towards *uncertain regions* of the state/action space

Outline

Towards an optimistic learning algorithm

- **Reminder** : Optimistic Bandit model



set of possible bandit models $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, \mu_4)$:

$$\mathcal{M}_t = \mathcal{I}_1(t) \times \mathcal{I}_2(t) \times \mathcal{I}_3(t) \times \mathcal{I}_4(t)$$

An optimistic bandit model is

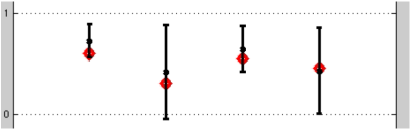
$$\boldsymbol{\mu}_t^+ \in \underset{\boldsymbol{\mu} \in \mathcal{M}_t}{\mathrm{argmax}} \ \mu^\star$$

➜ the best arm in $\boldsymbol{\mu}_t^+$ is $A_t = \underset{a \in \mathcal{A}}{\text{argmax}} \ \mathrm{UCB}_a(t)$

(arm selected by UCB)

Towards an optimistic learning algorithm

- **Reminder** : Optimistic Bandit model



set of possible bandit models $\boldsymbol{\mu} = (\mu_1, \mu_2, \mu_3, \mu_4)$:

$$\mathcal{M}_t = \mathcal{I}_1(t) \times \mathcal{I}_2(t) \times \mathcal{I}_3(t) \times \mathcal{I}_4(t)$$

An optimistic bandit model is

$$\boldsymbol{\mu}_t^+ \in \underset{\boldsymbol{\mu} \in \mathcal{M}_t}{\operatorname{argmax}} \max_a \mu_a$$

➜ the best arm in $\boldsymbol{\mu}_t^+$ is $A_t = \underset{a \in \mathcal{A}}{\text{argmax}}\ \mathrm{UCB}_a(t)$

(arm selected by UCB)

Towards an optimistic learning algorithm

- **Extension** : Optimistic Markov Decision Process

  set of possible MDPs $\boldsymbol{M} = \langle \mathcal{S}, \mathcal{A}, r, p \rangle$:

$$\mathcal{M}_t = \{\langle \mathcal{S}, \mathcal{A}, r, p \rangle : r, p \in \mathcal{B}_t^r \times \mathcal{B}_t^p\}$$

An optimistic Markov Decision Process is

$$\boldsymbol{M}_t^+ \in \operatorname*{argmax}_{\boldsymbol{M} \in \mathcal{M}_t} V_{\boldsymbol{M}}^\star(s_1)$$

→ an optimal policy in $\boldsymbol{M}_t^+$ is such that

$$\pi_t^+ \in \operatorname*{argmax}_{\pi} \max_{\boldsymbol{M} \in \mathcal{M}_t} V_{\boldsymbol{M}}^\pi(s_1)$$

Challenges

1. How to construct the set $\mathcal{M}_t$ of possible MDPs ?
2. How to numerically compute π_t^+ ?

Towards an optimistic learning algorithm

- **Extension** : Optimistic Markov Decision Process

$$\text{set of possible MDPs} \quad \boldsymbol{M} = \langle \mathcal{S}, \mathcal{A}, r, p \rangle :$$

$$\mathcal{M}_t = \{\langle \mathcal{S}, \mathcal{A}, r, p \rangle : r, p \in \mathcal{B}_t^r \times \mathcal{B}_t^p\}$$

An optimistic Markov Decision Process is

$$\boldsymbol{M}_t^+ \in \operatorname*{argmax}_{\boldsymbol{M} \in \mathcal{M}_t} \max_{\pi} V_{\boldsymbol{M}}^{\pi}(s_1)$$

→ an optimal policy in $\boldsymbol{M}_t^+$ is such that

$$\pi_t^+ \in \operatorname*{argmax}_{\pi} \max_{\boldsymbol{M} \in \mathcal{M}_t} V_{\boldsymbol{M}}^{\pi}(s_1)$$

Challenges

1. How to construct the set $\mathcal{M}_t$ of possible MDPs ?
2. How to numerically compute π_t^+ ?

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \Big\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a) \Big\}$$

Idea : build individual confidence regions

- on the average reward $r(s,a)$: $\mathcal{B}_t^r(s,a) \subseteq \mathbb{R}$
- on the transition probability vector $p(\cdot|s,a)$: $\mathcal{B}_t^p(s,a) \subseteq \Delta(\mathcal{S})$

that rely on the empirical estimates

$$\hat{r}_t(s,a) = \frac{1}{n_t(s,a)} \sum_{i=1}^{n_t(s,a)} r[i] \quad \text{and} \quad \hat{p}_t(s'|s,a) = \frac{n_t(s,a,s')}{n_t(s,a)}$$

$n_t(s,a)$: number of visits of (s,a) until episode t
$n_t(s,a,s')$: number of times s' was the next state when the transition (s,a) was performed until episode t

Goal : $\mathbb{P}_{\boldsymbol{M}} \left(\boldsymbol{M} \in \mathcal{M}_t \right)$ is close to 1

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \Big\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a) \Big\}$$

Idea : build individual confidence regions

- on the average reward $r(s,a)$: $\mathcal{B}_t^r(s,a) \subseteq \mathbb{R}$

Assuming bounded rewards,

$$\mathcal{B}_t^r(s,a) = \left[\hat{r}_t(s,a) - \sqrt{\frac{\ln(4(n_t(s,a))^2/\delta)}{2n_t(s,a)}} ; \hat{r}_t(s,a) + \sqrt{\frac{\ln(4(n_t(s,a))^2/\delta)}{2n_t(s,a)}} \right]$$

satisfies

$$\mathbb{P}\Big(\exists t \in \mathbb{N} : r(s,a) \notin \mathcal{B}_t^r(s,a) \Big) \leq \delta.$$

(Hoeffding inequality + union bound)

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \Big\{\langle \mathcal{S}, \mathcal{A}, r, p\rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a)\Big\}$$

Idea : build individual confidence regions

- on the average reward $r(s,a)$: $\mathcal{B}_t^r(s,a) \subseteq \mathbb{R}$

Assuming bounded rewards,

$$\mathcal{B}_t^r(s,a) = \Big[\hat{r}_t(s,a) - \beta_t^r(s,a); \hat{r}_t(s,a) + \beta_t^r(s,a)\Big]$$

satisfies

$$\mathbb{P}\Big(\exists t \in \mathbb{N} : r(s,a) \notin \mathcal{B}_t^r(s,a)\Big) \le \delta.$$

(Hoeffding inequality + union bound)

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \Big\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a) \Big\}$$

Idea : build individual confidence regions

- on the transition probability vector $p(\cdot|s,a)$: $\mathcal{B}_t^p(s,a) \subseteq \Delta(\mathcal{S})$

$$\mathcal{B}_t^p(s,a) = \left\{ p(\cdot|s,a) \in \Delta(\mathcal{S}) : \|p(\cdot|s,a) - \hat{p}_t(\cdot|s,a)\|_1 \leq C \sqrt{\frac{S \ln(n_t(s,a)/\delta)}{n_t(s,a)}} \right\}$$

satisfies

$$\mathbb{P}\Big(\exists t \in \mathbb{N} : p(\cdot|s,a) \notin \mathcal{B}_t^p(s,a) \Big) \leq \delta.$$

(Freedman inequality + union bound)
[Auer et al., 2008]

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \Big\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a) \Big\}$$

Idea : build individual confidence regions

- on the transition probability vector $p(\cdot|s,a)$: $\mathcal{B}_t^p(s,a) \subseteq \Delta(\mathcal{S})$

$$\mathcal{B}_t^p(s,a) = \left\{ p(\cdot|s,a) \in \Delta(\mathcal{S}) : \left\| p(\cdot|s,a) - \hat{p}_t(\cdot|s,a) \right\|_1 \leq \beta_t^p(s,a) \right\}$$

satisfies

$$\mathbb{P}\Big(\exists t \in \mathbb{N} : p(\cdot|s,a) \notin \mathcal{B}_t^p(s,a) \Big) \leq \delta.$$

(Freedman inequality + union bound)
[Auer et al., 2008]

Step 1 : Constructing $\mathcal{M}_t$

$$\mathcal{M}_t = \left\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}_t^r(s,a), p(\cdot|s,a) \in \mathcal{B}_t^p(s,a) \right\}$$

$$\begin{aligned} \mathcal{B}_t^r(s,a) &= \left[\hat{r}_t(s,a) - \beta_t^r(s,a); \hat{r}_t(s,a) + \beta_t^r(s,a) \right] \\ \mathcal{B}_t^p(s,a) &= \left\{ p(\cdot|s,a) \in \Delta(\mathcal{S}) : \|p(\cdot|s,a) - \hat{p}_t(\cdot|s,a)\|_1 \leq \beta_t^p(s,a) \right\} \end{aligned}$$

with exploration bonuses :

$$\begin{aligned} \beta_t^r(s,a) &\propto \sqrt{\frac{\ln(n_t(s,a)/\delta)}{n_t(s,a)}} \\ \beta_t^p(s,a) &\propto \sqrt{\frac{S \ln(n_t(s,a)/\delta)}{n_t(s,a)}} \end{aligned}$$

Step 2 : Optimistic Value Iteration

Goal : Approximate $\pi^+ \in \operatorname{argmax}_{\pi} \max_{\boldsymbol{M} \in \mathcal{M}} V^{\pi}_{\boldsymbol{M}}$ for a set of MDPs

$$\mathcal{M} = \left\{ \langle \mathcal{S}, \mathcal{A}, r, p \rangle : \forall (s,a) \in \mathcal{S} \times \mathcal{A}, r(s,a) \in \mathcal{B}^r(s,a), p(\cdot|s,a) \in \mathcal{B}^p(s,a) \right\}$$

Recall the optimal solution for a fixed MDP : $\pi^\star_h = \text{greedy}(Q^\star_h)$ where

$$Q^\star_h(s,a) = r(s,a) + \sum_{s'} p(s'|s,a) \max_b Q^\star_{h+1}(s',b)$$

➜ $\pi^+_h = \text{greedy}(Q^+_h)$ where

$$Q^+_h(s,a) = \max_{(r,p) \in \mathcal{M}} \left[r(s,a) + \sum_{s'} p(s'|s,a) \max_b Q^+_{h+1}(s',b) \right]$$

Step 2 : Optimistic Value Iteration

$$Q_h^+(s,a) = \max_{(r,p)\in\mathcal{B}^r(s,a)\times\mathcal{B}^p(s,a)} \left[r(s,a) + p(\cdot|s,a)^\top \underbrace{\left(\max_b Q_{h+1}^+(s',b)\right)_{s'\in\mathcal{S}}}_{V_{h+1}^+} \right]$$

$$= \max_{r\in\mathcal{B}^r(s,a)} r + \max_{p\in\mathcal{B}^p(s,a)} p^\top V_{h+1}^+$$

$$= \hat{r}_t(s,a) + \beta_t^r(s,a) + \max_{p\in\mathcal{B}^p(s,a)} p^\top V_{h+1}^+$$

$$= \hat{r}_t(s,a) + \beta_t^r(s,a) + \hat{p}_t(\cdot|s,a)^\top V_{h+1}^+ + \max_{p\in\mathcal{B}^p(s,a)} (p - \hat{p}_t(\cdot|s,a))^\top V_{h+1}^+$$

$$\leq \hat{r}_t(s,a) + \beta_t^r(s,a) + \hat{p}_t(\cdot|s,a)^\top V_{h+1}^+ + \max_{p\in\mathcal{B}^p(s,a)} \|p - \hat{p}_t(\cdot|s,a)\|_1 \|V_{h+1}^+\|_\infty$$

$$= \hat{r}_t(s,a) + \beta_t^r(s,a) + \hat{p}_t(\cdot|s,a)^\top V_{h+1}^+ + \beta_t^p(s,a)(H-h)r_{\max}$$

$$= \hat{r}_t(s,a) + \underbrace{[\beta_t^r(s,a) + \beta_t^p(s,a)(H-h)r_{\max}]}_{\text{exploration bonus}} + \hat{p}_t(\cdot|s,a)^\top V_{h+1}^+$$

Optimistic algorithm

A family of algorithms

An **optimistic algorithm** uses in episode $t+1$ the exporation policy $\pi_h^{t+1} = \text{greedy}\left(\overline{Q}_h\right)$ where $\overline{Q}_h(s,a)$ is an optimistic Q-value function

$$\overline{Q}_h(s,a) = \hat{r}_t(s,a) + \beta_t(s,a) + \sum_{s' \in \mathcal{S}} \hat{p}_t(s'|s,a) \max_b \overline{Q}_{h+1}(s',b)$$

where $\beta_t(s,a)$ is some exploration bonus.

From the previous calculation, one can propose

$$\beta_t(s,a) = \beta_t^r(s,a) + C\beta_t^p(s,a) \simeq \sqrt{\frac{\ln(n_t(s,a))}{n_t(s,a)}} + C\sqrt{\frac{S\ln(n_t(s,a))}{n_t(s,a)}}$$

➜ $\beta_t(s,a)$ scales in $1/\sqrt{n_t(s,a)}$ where $n_t(s,a)$ is the number of previous visits to (s,a).

Optimistic algorithm

A family of algorithms

An **optimistic algorithm** uses in episode $t+1$ the exporation policy $\pi_h^{t+1} = \text{greedy}\left(\overline{Q}_h\right)$ where $\overline{Q}_h(s,a)$ is an optimistic Q-value function

$$\overline{Q}_h(s,a) = \hat{r}_t(s,a) + \beta_t(s,a) + \sum_{s' \in \mathcal{S}} \hat{p}_t(s'|s,a) \max_b \overline{Q}_{h+1}(s',b)$$

where $\beta_t(s,a)$ is some exploration bonus.

- An example of optimistic algorithm in the episodic setting : UCB-VI [Azar et al., 2017]
- Optimistic algorithms were first proposed in the more complex average-reward MDPs : UCRL [Auer et al., 2008]

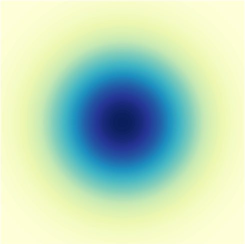
Regret

UCB-VI achieves $R_T = \mathcal{O}(\sqrt{H^2 SAT})$ w.h.p.

Outline

Posterior Sampling for RL

Bayesian assumption : $\boldsymbol{M}$ is drawn from some prior distribution ν_0.



$$\nu_t \in \Delta(\mathcal{M}) : \quad \text{posterior distribution over the set of MDPs}$$

Optimism	**Posterior Sampling**
Set of possible MDPs	Posterior distribution over MDPs
Compute the optimistic MDP	Sample from the posterior distribution

Posterior Sampling for Episodic RL

Algorithm 1: PSRL

Input : Prior distribution ν_0

for $t = 1, 2, \dots$ **do**

  $s_1 \sim \rho$ \\\\ get the starting state of episode t

  Sample $\widetilde{M}_t \sim \nu_{t-1}$ \\\\ sample an MDP from the current posterior distribution

  Compute $\tilde{\pi}^t$ an optimal policy for $\widetilde{M}_t$

  for $h = 1, \dots, H$ **do**

    $a_h = \tilde{\pi}^t_h(s_h)$ \\\\ choose next action according to $\tilde{\pi}^t$

    $r_h, s_{h+1} = \mathrm{step}(s_h, a_h)$

  end

  Compute ν_t based on ν_{t-1} and $\{(s_h, a_h, r_h, s_{h+1})\}_{h=1}^H$

end

[Strens, 2000, Osband et al., 2013]

Outline

Limitations of optimistic approaches

An important message from optimistic approaches :

➜ Do not only trust the estimated MDP $\hat{M}_t$, but take into account the uncertainty in the underlying estimate

$$
\begin{aligned}
\mathcal{B}_t^r(s,a) &= \left[\hat{r}_t(s,a) - \beta_t^r(s,a); \hat{r}_t(s,a) + \beta_t^r(s,a)\right] \\
\mathcal{B}_t^p(s,a) &= \left\{p(\cdot|s,a) \in \Delta(\mathcal{S}) : \|p(\cdot|s,a) - \hat{p}_t(\cdot|s,a)\|_1 \leq \beta_t^p(s,a)\right\}
\end{aligned}
$$

expressed by exploration bonuses scaling in $\sqrt{\frac{1}{n_t(s,a)}}$ where $n_t(s,a)$ is the count (=number of visits) of (s,a).

Scaling for large state action spaces ?

- each state action pair may be visited only very little...
- UCB-VI is quite inefficient in practice for large state-spaces (efficient, continuous variants is an active research direction)

A heuristic : count-based exploration

General principle

1. Estimate a "proxi" for the number of visits of a state $\tilde{n}_t(s)$
2. Add an exploration bonus directly to the collected rewards :

$$r_t^+ = r_t + c\sqrt{\frac{1}{\tilde{n}_t(s_t)}}$$

3. Run any DeepRL algorithm on

$$\mathcal{D} = \bigcup_t \left\{ \left(s_t, a_t, r_t^+, s_{t+1} \right) \right\}$$

A heuristic : count-based exploration

General principle

1. Estimate a “proxi” for the number of visits of a state $\tilde{n}_t(s)$
2. Add an exploration bonus directly to the collected rewards :

$$r_t^+ = r_t + c\sqrt{\frac{1}{\tilde{n}_t(s_t)}}$$

3. Run any DeepRL algorithm on

$$\mathcal{D} = \bigcup_t \left\{(s_t, a_t, r_t^+, s_{t+1})\right\}$$

Example of pseudo-counts :

- use a hash function, e.g. $\phi : \mathcal{S} \to \{-1, 1\}^k$
  $n(\phi(s_t)) \leftarrow n(\phi(s_t)) + 1$
  (possibly learn a good hash function)

[Tang et al., 2017]

Limitations of Posterior Sampling

An important message from posterior sampling :

➜ Adding some noise to the estimated MDP $\hat{M}_t$ is helpful !

$$\begin{aligned} \tilde{r}_t(s,a) &= \hat{r}_t(s,a) + \epsilon_t(s,a) \\ \tilde{p}_t(s'|s,a) &= \hat{p}_t(\cdot|s,a) + \epsilon'_t(s,a). \end{aligned}$$

Scaling for large state action spaces ?

- maintaining independent posterior over all state action rewards and transitions can be costly
- more sophisticated prior distributions encoding some structure and the associated posteriors can be hard to sample from

➜ use other type of (non-Bayesian) randomized exploration ?
Noisy Networks [Fortunato et al., 2017]
Bootstrap DQN [Osband et al., 2016] ...

Outline

Monte-Carlo Tree Search

MCTS is a family of methods that use possibly random exploration to explore the tree of possible next states.

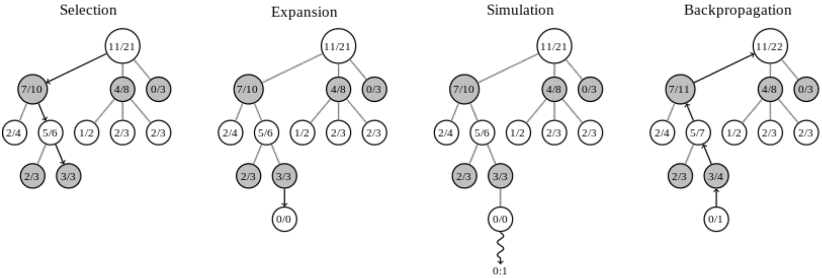


FIGURE – An generic MCTS algorithm illustrated for a game

The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

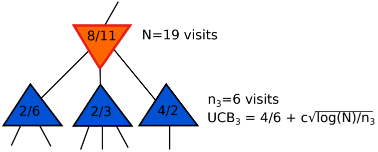
UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MAX node s (= root player move), select an action

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmax}} \ \frac{S(s,a)}{N(s,a)} + c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning



The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

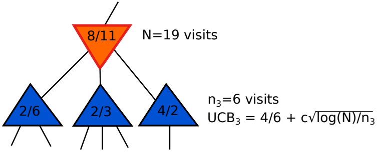
UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MIN node s (= adversary move), select an action

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmin}} \ \frac{S(s,a)}{N(s,a)} - c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning



The UCT algorithm

Bandit-Based Monte-Carlo planning : to select a path in the tree, run a bandit algorithm each time a children (next action) needs to be selected

UCT = UCB for Trees [Kocsis and Szepesvári, 2006]

UCT in a Game Tree

In a MAX node s (= root player move), select an action

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmax}} \ \frac{S(s,a)}{N(s,a)} + c\sqrt{\frac{\ln\left(\sum_b N(s,b)\right)}{N(s,a)}}$$

$N(s,a)$: number of visits of (s,a)
$S(s,a)$: number of visits of (s,a) ending with the root player winning

When a leaf (or some maximal depth) is reached :

- a playout is performed (play the game until the end with a simple heuristic, or produce a random evaluation of the leaf position)
- the outcome of the playout (typically 1/0) is stored in all the nodes visited in the previous trajectory

The UCT algorithm

- first good AIs for Go where based on variants on UCT
- it remains a heuristic (no sample complexity guarantees, parameter c fined-tuned for each application)
- many variants have been proposed

[Browne et al., 2012]

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Selection step : in some state s, choose the next action to be

$$\underset{a \in \mathcal{C}(s)}{\operatorname{argmax}} \left[\frac{S(s,a)}{N(s,a)} + c \times P(s,a) \frac{\sqrt{N(s)}}{1 + N(s,a)} \right]$$

for some (fine-tuned) constant c.

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Expansion step : once a leaf s_L is reached, compute $(\boldsymbol{p}, v) = f_\theta(s_L)$.

- Set v to be the value of the leaf
- For all possible next actions b :
  - ➜ initialize the count $N(s_L, b) = 0$
  - ➜ initialize the prior probability $P(s_L, b) = \boldsymbol{p}_b$ (possibly add some noise)

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network
$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Back-up step : for all ancestor s_t, a_t in the trajectory that end in leaf s_L,

$$\begin{aligned} N(s_t, a_t) &\leftarrow N(s_t, a_t) + 1 \\ S(s_t, a_t) &\leftarrow S(s_t, a_t) + v \end{aligned}$$

Alpha Zero

AlphaZero learns a good policy by using a MCTS algorithm guided by a neural network

$\neq$ pure play-out based MCTS

Input

A neural network predicting a policy $\boldsymbol{p} \in \Delta(\mathcal{A})$ and a value $v \in \mathbb{R}$ from the current state s : $(\boldsymbol{p}, v) = f_\theta(s)$.

The MCTS algorithm maintains for each visited state/action the counts and cumulated values + a vector of prior action probabilities :

$$\{N(s,a), S(s,a), P(s,a)\}$$

Output of the planning algorithm ? select an action a at random according to

$$\pi(a) = \frac{N(s_0,a)^{1/\tau}}{\sum_b N(s_0,b)^{1/\tau}}$$

for some (fine-tuned) temperature τ.

Training the neural network

- In AlphaGo, f_θ was trained on a database of games played by human
- In AlphaZero, the network is trained using only self-play

[Silver et al., 2016, Silver et al., 2017]

Let θ be the current parameter of the network $(\boldsymbol{p}, v) = f_\theta(s_L)$.

1. generate N games where each player uses MCTS(θ) to select the next action a_t (and output a probability over actions π_t)

$$\mathcal{D} = \bigcup_{i=1}^{\text{Nb games}} \left\{ \left(s_t, \pi_t, \pm r_{T_i}\right) \right\}_{i=1}^{T_i}$$

T_i : length of game i, $r_{T_i} \in \{-1, 0, 1\}$ outcome of game i for one player

2. Based on a sub-sample of $\mathcal{D}$, train the neural network using stochastic gradient descent on the loss function

$$L(s, \boldsymbol{\pi}, z; \boldsymbol{p}, v) = (z - v)^2 - \boldsymbol{\pi}^\top \ln(\boldsymbol{p}) + c\|\theta\|^2$$

A nice actor-critic architecture

AlphaZero alternates between

- **The actor** : MCTS(θ)
  generates trajectories guided by the network f_θ but still exploring

➜ act as a policy improvement
  ($N = 25000$ games played, each call to MCTS uses 1600 simulations)

- **The critic** : neural network f_θ
  updates θ based on trajectories followed by the critic

➜ evaluate the actor's policy

Summary

Bandit tools can be useful in more realistic, contextual models

Bandits tools are useful for Reinforcement Learning :

- UCRL, PSRL : bandit-based exploration for tabular MDPs
- ... that can motivate "deeper" heuristics

Bandit tools lead to big success in Monte-Carlo planning

- ... without proper sample complexity guarantees

➜ Unifying theory and practice is a big challenge in RL !

Perspective : bandit tools are also useful beyond RL (i.e. with no rewards to maximize) : best arm identification, black box optimization...

Agrawal, S. and Goyal, N. (2013).
Thompson sampling for contextual bandits with linear payoffs.
In *International conference on machine learning*, pages 127–135. PMLR.

Auer, P., Jaksch, T., and Ortner, R. (2008).
Near-optimal regret bounds for reinforcement learning.
Advances in neural information processing systems, 21.

Azar, M. G., Osband, I., and Munos, R. (2017).
Minimax regret bounds for reinforcement learning.
In *International Conference on Machine Learning*, pages 263–272. PMLR.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012).
A survey of monte carlo tree search methods.
IEEE Transactions on Computational Intelligence and AI in games, 4(1) :1–43.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. (2017).
Noisy networks for exploration.
arXiv preprint arXiv :1706.10295.

Kocsis, L. and Szepesvári, C. (2006).
Bandit based monte-carlo planning.
In *European conference on machine learning*, pages 282–293. Springer.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016).
Deep exploration via bootstrapped dqn.
Advances in neural information processing systems, 29.

Osband, I., Russo, D., and Van Roy, B. (2013).
(more) efficient reinforcement learning via posterior sampling.
Advances in Neural Information Processing Systems, 26.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016).
Mastering the game of go with deep neural networks and tree search.
nature, 529(7587) :484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017).
Mastering the game of go without human knowledge.
nature, 550(7676) :354–359.

Strens, M. (2000).
A bayesian framework for reinforcement learning.
In *ICML*, volume 2000, pages 943–950.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. (2017).
exploration : A study of count-based exploration for deep reinforcement learning.
Advances in neural information processing systems, 30.