Reinforcement Learning: Multi-Armed Bandits

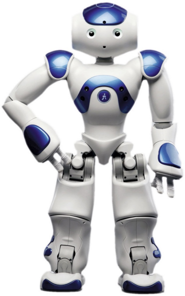
Thomas Bonald

2019 – 2020

TELECOM Paris
IP PARIS

Reinforcement Learning

- Learning by **trial and error**
- Inspired by the behavior of animals (including humans!)
- The **exploration-exploitation** trade-off
- Many applications: robotics, games, advertising, content recommendation, medicine, etc.



Outline

1. Multi-armed bandits
2. Main algorithms
3. Performance analysis
4. Extensions
5. A case study: AlphaGo

Multi-armed bandits

- A class of RL problems where the agent does **not** modify its environment
- At time $t = 1, 2, \ldots$, the agent selects an **action** a_t in some **finite** set A and receives some **reward** r_t
- The rewards of action $a \in A$ are i.i.d. with some **unknown** distribution $p(\cdot|a)$, with expectation

$$q(a) = \mathrm{E}(r|a) = \sum_r rp(r|a)$$

- The objective is to find and to **exploit** the best action(s) on observing the rewards

Example: A/B testing

- Objective: find the best version of a Web site
- Action = show version A or B
- Reward (binary) = click / no click

Example: Obama 2008 campaign

Time horizon

- The objective is to maximize the **cumulative reward** over some (possibly unknown) time horizon T:

$$\sum_{t=1}^{T} r_t$$

- If action a is always selected, the cumulative reward is approximately $Tq(a)$ for large T
- Maximum reward (per action):

$$q^\star = \max_a q(a)$$

- Best action(s)

$$a^\star = \arg\max_a q(a)$$

Performance metrics

1. **Cumulative regret** (gap to optimal reward)

$$R = q^\star T - \sum_{t=1}^{T} r_t$$

2. **Precision** (proportion of optimal actions)

$$P = \frac{1}{T} \sum_{t=1}^{T} 1_{\{a_t = a^\star\}}$$

Performance metrics (in expectation)

Let $N_t(a)$ be # of times action a has been selected up to time t

1. **Cumulative regret** (expected gap to optimal reward)

$$\begin{aligned}\mathrm{E}(R) &= q^\star T - \sum_{t=1}^{T} \mathrm{E}(r_t) \\ &= \sum_{a \in A} (q^\star - q(a)) \mathrm{E}(N_T(a))\end{aligned}$$

2. **Precision** (expected proportion of optimal actions)

$$\begin{aligned}\mathrm{E}(P) &= \frac{1}{T} \sum_{t=1}^{T} P(a_t = a^\star) \\ &= \frac{\mathrm{E}(N_T(a^\star))}{T}\end{aligned}$$

Example

Action	a	b	c
Expected reward	1	9	10

Policy A

Action	a	b	c
Distribution	10%	40%	50%

Regret (per action) $= 1.3$, Precision $= 0.5$

Policy B

Action	a	b	c
Distribution	20%	20%	60%

Regret (per action) $= 2$, Precision $= 0.6$

Efficiency

- Efficient algorithm = **sublinear regret**

$$\frac{\mathrm{E}(R)}{T} \to 0 \quad \text{when} \quad T \to +\infty$$

- Since

$$\mathrm{E}(R) = \sum_{a \in A} (q^\star - q(a)) \mathrm{E}(N_T(a))$$

this implies

$$\forall a \neq a^\star, \quad \frac{\mathrm{E}(N_T(a))}{T} \to 0$$

and

$$P \to 1$$

Outline

1. Multi-armed bandits
2. **Main algorithms**
3. Performance analysis
4. Extensions
5. A case study: AlphaGo

Variables

Keep for each action a:

- $R(a) =$ cumulative reward of action a
- $N(a) =$ number of selections of action a

$\rightarrow$ estimation of the value of action a:

$$Q(a) = \frac{R(a)}{N(a)}$$

Start for each action a with:

- $R(a) \leftarrow 0$
- $N(a) \leftarrow 0$

$\rightarrow$ the initial value of $Q(a)$ is undefined!

A first algorithm

Greedy algorithm

Repeat:

- $a \leftarrow \arg\max_a Q(a)$ (random tie breaking)
- $r \leftarrow \text{reward}(a)$
- $R(a) \leftarrow R(a) + r$
- $N(a) \leftarrow N(a) + 1$
- $Q(a) \leftarrow \frac{R(a)}{N(a)}$

Note: The initial value of $Q(a)$ plays a key role!
$\rightarrow$ Be **optimistic** in case of uncertainty

A second algorithm

ε-greedy algorithm

Parameter: ε

Repeat:

- $a \leftarrow \begin{cases} \text{random action} & \text{with probability } \varepsilon \\ \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \end{cases}$
- $r \leftarrow \text{reward}(a)$
- $R(a) \leftarrow R(a) + r$
- $N(a) \leftarrow N(a) + 1$
- $Q(a) \leftarrow \frac{R(a)}{N(a)}$

An adaptive algorithm

Adaptive-greedy algorithm

Parameter: c

Repeat for $t = 1, 2, \ldots$

- $\varepsilon \leftarrow \frac{c}{c+t}$
- $a \leftarrow \begin{cases} \text{random action} & \text{with probability } \varepsilon \\ \arg\max_a Q(a) & \text{with probability } 1 - \varepsilon \end{cases}$
- $r \leftarrow \text{reward}(a)$
- $R(a) \leftarrow R(a) + r$
- $N(a) \leftarrow N(a) + 1$
- $Q(a) \leftarrow \frac{R(a)}{N(a)}$

Upper confident bound

Idea $=$ **bonus** for uncertainty

UCB algorithm

Parameter: c

Repeat for $t = 1, 2, \ldots$

- $a \leftarrow \arg\max_a \left(Q(a) + c\sqrt{\frac{\log t}{N(a)}}\right)$
- $r \leftarrow \text{reward}(a)$
- $R(a) \leftarrow R(a) + r$
- $N(a) \leftarrow N(a) + 1$
- $Q(a) \leftarrow \frac{R(a)}{N(a)}$

(Auer, Cesa-Bianchi & Fischer 2002)

Bayesian algorithm

Idea = replace uncertainty by... **randomness!**

Thompson sampling

Initialize, for all actions a:

- $P(a) \leftarrow$ prior on $Q(a)$

Repeat:

- for all actions a, $Q(a) \leftarrow \text{sample}(P(a))$
- $a \leftarrow \arg\max_a Q(a)$
- $r \leftarrow \text{reward}(a)$
- $P(a) \leftarrow \text{update}(r)$

Proposed by Thompson in... 1933!

Thompson sampling: binary rewards

- Prior = **uniform distribution**

$$p(q) = 1_{(0,1)}(q)$$

- Writing

$$p(r|q) = q^r(1-q)^{1-r}, \quad r = 0, 1,$$

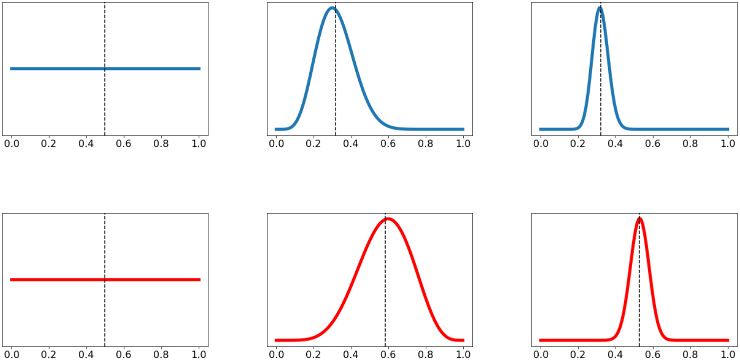
the **posterior distribution** follows from Bayes' rule:

$$\begin{aligned} p(q|r_1,\ldots,r_N) &= \frac{p(r_1,\ldots,r_N|q)p(q)}{p(r_1,\ldots,r_N)} \\ &\propto q^{r_1+\ldots+r_N}(1-q)^{N-(r_1+\ldots+r_N)} \end{aligned}$$

- This is a **Beta distribution** with parameters $s+1, N-s+1$, with $s = r_1 + \ldots + r_N$

Example

- True parameters $= 0.3$ and 0.5
- Beta distribution after $N = 0, 10, 100$ tries of each:



Thompson sampling: normal rewards

- Prior = **standard normal distribution**

$$p(q) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}q^2}$$

- Since

$$p(r|q) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(r-q)^2},$$

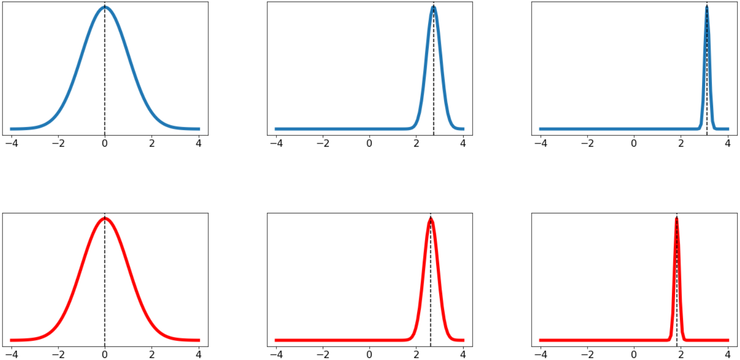
  the **posterior distribution** follows from Bayes' rule:

$$\begin{aligned} p(q|r_1, \ldots, r_N) &= \frac{p(r_1, \ldots, r_N|q)p(q)}{p(r_1, \ldots, r_N)} \\ &\propto e^{-\frac{N+1}{2}\left(q - \frac{1}{N+1}(r_1 + \ldots + r_N)\right)^2} \end{aligned}$$

- This is a **normal distribution** with mean $\frac{1}{N+1}(r_1 + \ldots + r_N)$ and variance $\frac{1}{N+1}$

Example

- True parameters $= 3$ and 2 (variance 1)
- Normal distribution after $N = 0, 10, 100$ tries:

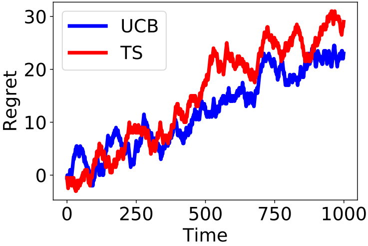


Outline

1. Multi-armed bandits
2. Main algorithms
3. **Performance analysis**
4. Extensions
5. A case study: AlphaGo

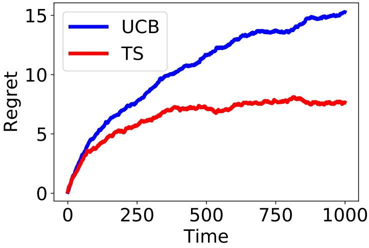
Example

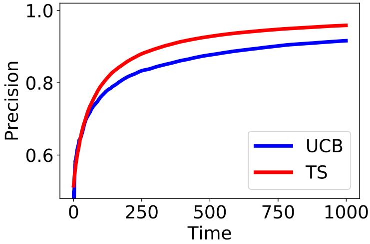
- Binary rewards
- Parameters = 0.3 and 0.5



Example

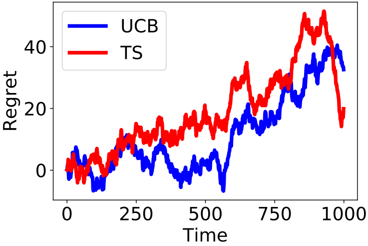
- Binary rewards
- Parameters $= 0.3$ and 0.5
- 200 independent runs





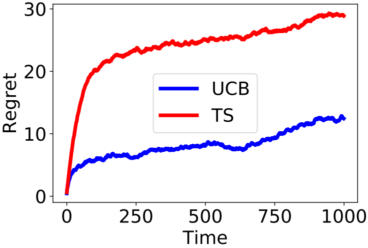
Example

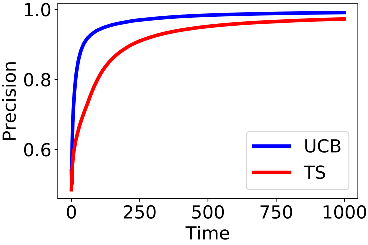
- Normal rewards
- Parameters = 2 and 3 (variance 1)



Example

- Normal rewards
- Parameters = 2 and 3 (variance 1)
- 200 independent runs





Efficiency of UCB and TS

- Expected regret:

$$\mathrm{E}(R) = O(\log T) \quad \to \textbf{sublinear}$$

- For **binary rewards**, the multiplicative constant is:

$$\sum_{a \neq a^\star} \frac{1}{q^\star - q(a)} \quad \text{for UCB}$$

$$\sum_{a \neq a^\star} \frac{q^\star - q(a)}{D(q(a)||q^\star)} \quad \text{for TS}$$

$\to$ highest cost due to **quasi-optimal** actions!

Lower bound (binary rewards)

- A fundamental bound valid for **any** algorithm with sublinear regret (Lai & Robbins 1985)
- For any suboptimal action $a \neq a^\star$,

$$\liminf_{T \to +\infty} \frac{N_T(a)}{\log T} \geq \frac{1}{D(q(a)||q^\star)}$$

- In particular,

$$\liminf_{T \to +\infty} \frac{\mathrm{E}(R)}{\log T} \geq \sum_{a \neq a^\star} \frac{q^\star - q(a)}{D(q(a)||q^\star)}$$

$\rightarrow$ TS is **optimal**

Outline

1. Multi-armed bandits
2. Main algorithms
3. Performance analysis
4. **Extensions**
5. A case study: AlphaGo

Contextual bandits

- Some **context** associated with each action (e.g., information on user for advertising)
- At time $t = 1, 2, \ldots$, choose **action** a_t based on **state** s_t (context) and get **reward** r_t that depends on both a_t and s_t
- Case of **linear bandits**:

$$q(a) =< \theta(a), s > \quad \theta(a), s \in \mathbb{R}^d$$

Need to learn both $\theta(a)$ for each action a and $a^\star$
LinUCB algorithm combines linear regression and UCB

Adaptive bandits

- Assume some **slowly varying** environment
- **Discounted UCB**: Select the action a_t maximizing:

$$Q(a) + c\sqrt{\frac{1}{N(a)} \log \frac{1-\gamma^t}{1-\gamma}}$$

then update for **all** actions:

$$\begin{aligned} R(a) &\leftarrow \gamma R(a) + r 1_{\{a_t = a\}} \\ N(a) &\leftarrow \gamma N(a) + 1_{\{a_t = a\}} \\ Q(a) &\leftarrow \frac{R(a)}{N(a)} \end{aligned}$$

and $\gamma < 1$ is the discount factor

Adversarial bandits

- Assume the rewards are **arbitrary** sequences, chosen by some **adversary**
- Regret is for the **worst-case** scenario (minimax regret):

$$\min_a \max_r \mathrm{E}(R)$$

where the regret is now

$$R = \max_a \sum_{t=1}^{T} r_{t,a} - \sum_{t=1}^{T} r_{t,a_t}$$

- **Exp3** (Exponential-weight algorithm for Exploration and Exploitation) $\rightarrow$ random selection of action a with prob. $\propto w(a)$, adapted through $w(a) \leftarrow w(a)e^{\eta r}$

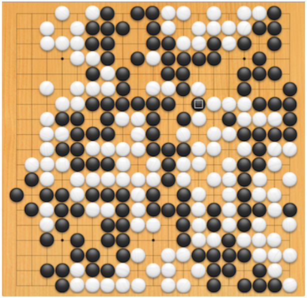
Outline

1. Multi-armed bandits
2. Main algorithms
3. Performance analysis
4. Extensions
5. **A case study: AlphaGo**

Brief history

Go has long been considered as the **ultimate game**, showing the superiority of humans over machines...

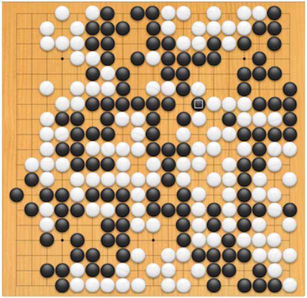
- In 2015, AlphaGo beats a human professional
- In 2016, **AlphaGo** beats the world champion Lee Sedol 4-1
- In 2017, **AlphaGo Zero** beats AlphaGo 100-0
- 2 months later, **AlphaZero** beats the world-champion prog. Stockfish (chess), Elmo (shogi) and AlphaGo Zero (go)



AlphaGo Zero

Based on **Monte-Carlo Tree Search** with

- **bandit algorithm** for opportunistic exploration
- **deep learning** for reward estimation



$3^{19\times19}$ states $\approx 10^{172}$

Monte-Carlo Tree Search

Built from each state (the **root** of the tree) to choose the action, through simulations (e.g., 1600 runs in AlphaGo)

Based on 4 steps:

- **Selection:** select a leaf starting from the root
  $\rightarrow$ bandit algorithm
- **Expansion:** expand this leaf with all possible actions
- **Simulation:** estimate the outcome of the game from the leaf
  $\rightarrow$ replaced by the neural net in AlphaGo
- **Backpropagation:** update the returns on the path from the leaf back to the root

Bandit algorithm in AlphaGo

Store in each **branch** of the tree:

- $R(s, a)$, cumulative reward of s after action a
- $N(s, a)$, number of selections of action a in state s
- $P(s, a)$, prior prob. of action a in state s
  (given by the neural net)

Select **action** a that maximizes $Q(s, a) + cU(s, a)$ with:

$$Q(s, a) = \frac{R(s, a)}{N(s, a)}, \quad U(s, a) = P(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)}$$

After 1600 runs, select for the root, say s_0, the action a with probability $\pi(a) \propto N(s_0, a)^{1/\tau}$ (for some parameter $\tau > 0$) and discard the tree except the **subtree** under action a

Neural network

Input = game state + short history

- black = $8 \times 19 \times 19$
- white = $8 \times 19 \times 19$
- total $\approx 6\,000$ bits

Output = moves (policy) + outcome (value)

- move probabilities $\in [0,1]^{19\times 19}$
- win probability $\in [0,1]$

Architecture = deep CNN

- 40 layers
- $\approx 184\,000$ neurons

Training the neural net

Self play

- The best current player (= best neural net) plays 25 000 games against itself
- Each game is stored, with the move prob. π and the outcome

Training $\rightarrow$ new player

- 1 000 epochs
- Each epoch $=$ 2 048 states from the last 500 000 games
- Loss $=$ KL on π $+$ MSE on outcome $\in \{0, 1\}$ $+$ regularization

Competition

- Play 400 games of best current player vs. new player
- Decide that the new player is the best player if it wins at least 55% of the games

References

Max Pagels
Practical AI for Business: Bandit algorithms (2017)

Émilie Kaufmann
PhD thesis (2014)

Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband and Zheng Wen A tutorial on Thompson Sampling (2018)

David Silver
Course on Reinforcement Learning
AlphaZero explained (2018)

David Foster
How to build your own AlphaZero AI using Python and Keras
with Python code (2018)