AI learns to act

Philippe Preux
philippe.preux@univ-lille.fr
SequeL

Artificial Intelligence learns to act.

~~Artificial~~ ~~Intelligence~~ learns to act.

Algorithms learn to Act.

Algorithms learn to act

Acting:

- turns out to making a series of decisions and put them into realization
- requires interaction between an acting agent and its environment
- when the agent takes a decision, it is based on previous interactions with its environment
- this is fundamentally a sequential process

Algorithms learn to act

Acting:

- turns out to making a series of decisions and put them into realization
- requires interaction between an acting agent and its environment
- when the agent takes a decision, it is based on previous interactions with its environment
- this is fundamentally a sequential process

Learning to act: there is uncertainty, stochasticity, in the environment and the agent has to learn by interacting with it.

Algorithms learn to act

Acting:

- turns out to making a series of decisions and put them into realization
- requires interaction between an acting agent and its environment
- when the agent takes a decision, it is based on previous interactions with its environment
- this is fundamentally a sequential process

Learning to act: there is uncertainty, stochasticity, in the environment and the agent has to learn by interacting with it.

Examples: an agent learning:

- to play a game
- to drive an autonomous vehicle
- to control a smart grid

Roadmap

- supervised learning in 5 minutes: all you need to know for the rest of the talk
- learning to act:
  - the bandit problem
  - the reinforcement learning problem
- outro

Some background on machine learning
5 minutes on supervised learning

- supervised learning is all about learning to predict a label given a data, and given a set of examples
- an example = (data, label)
- a datum = set of attributes
- a label =
  - a class (nominal value) $\rightsquigarrow$ supervised classification problem
  - a rank (ordinal value) $\rightsquigarrow$ ranking problem
  - a real number $\rightsquigarrow$ regression problem
  - a subset of nominal values $\rightsquigarrow$ multi-label supervised classfication
  - a text, *e.g.* text captioning
  - a set of real numbers (a vector, a matrix, a tensor), *e.g.* bounding box regression
  - any data structure (sequence, tree, graph, ...) $\rightsquigarrow$ structured output prediction problem
- we assume there exists a statistical model giving the (probability of) a label given a data (but we don't know it).

Some background on machine learning

5 minutes on supervised learning

A lof of different methods:

- k nearest neighbors
- decision tree
- Bayesian method
- multi-layer perceptron (= shallow or deep neural network)
- support vector machines
- ensemble methods: boosting, random forests, ...
- ...

Some background on machine learning

5 minutes on supervised learning

- During this talk, I will mainly need to solve regression problems.
- This is a tool for me.
- We assume we know how to solve it.
- However, it is not so obvious, and more research is still required on regression problems.
- Overfitting issues.

The bandit problem

The bandit problem
Setting

- K arms/alternatives, each with an unkown reward law ν_k
- Iteratively: pull an arm and observe the consequences
- Goal:
  - gather as much rewards as possible, or
  - find the best arm
- Setting: finite horizon (known or not), or infinite.
- (there are other, closely related, settings)

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:

$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1{=}5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next?

The bandit problem
Some strategies

$K = 4$ arms, $t = 26$ pulls:




$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next? Many strategies:

- ϵ-greedy:
- ϵ-decreasing greedy:
- proportional:
- softmax:
- ...

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:




$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next? Many strategies:

- ϵ-greedy: Pull $\arg\max_k \hat{\mu}_k$ with probability $1 - \epsilon$ or pick an arm at random.
- ϵ-decreasing greedy:
- proportional:
- softmax:
- ...

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:




$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6$ $\hat{\mu}_2 = \frac{1}{4} = .25$ $\hat{\mu}_3 = \frac{6}{10} = .6$ $\hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next? Many strategies:

- ϵ-greedy: Pull $\arg\max_k \hat{\mu}_k$ with probability $1 - \epsilon$ or pick an arm at random.
- ϵ-decreasing greedy: same with a diminishing ϵ (*e.g.* $\epsilon = \frac{1}{\sqrt{(t)}}$)
- proportional:
- softmax:
- ...

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:




$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next? Many strategies:

- ϵ-greedy: Pull $\arg\max_k \hat{\mu}_k$ with probability $1 - \epsilon$ or pick an arm at random.
- ϵ-decreasing greedy: same with a diminishing ϵ (*e.g.* $\epsilon = \frac{1}{\sqrt{(t)}}$)
- proportional: Pull arm k with probability proportional to $\hat{\mu}_k$.
- softmax:
- ...

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:




$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

Which arm do you pull next? Many strategies:

- ϵ-greedy: Pull $\arg\max_k \hat{\mu}_k$ with probability $1 - \epsilon$ or pick an arm at random.
- ϵ-decreasing greedy: same with a diminishing ϵ (*e.g.* $\epsilon = \frac{1}{\sqrt{(t)}}$)
- proportional: Pull arm k with probability proportional to $\hat{\mu}_k$.
- softmax: Pull arm k with probability proportional to $e^{\frac{\hat{\mu}_k}{\tau}}$ with $\tau > 0$ diminishing.
- ...

The bandit problem

Key notion of performance: the regret

We evaluate the performance in terms of the regret:

- $r_t =$ what you get - what you could have got if choosing the best arm (on average)

The bandit problem

Key notion of performance: the regret

We evaluate the performance in terms of the regret:

- r_t = what you get - what you could have got if choosing the best arm (on average)
- cumulated regret: $R_T = \sum_{t=1}^{t=T} r_t$

The bandit problem

Key notion of performance: the regret

We evaluate the performance in terms of the regret:

- r_t = what you get - what you could have got if choosing the best arm (on average)
- cumulated regret: $R_T = \sum_{t=1}^{t=T} r_t$
- $R_T = T\mu^* - \sum_{t=1}^{t=T} \mathbb{E}[r_t] = T\mu^* - \sum_{t=1}^{t=T} \mu_{k_t}$
  where μ^* is the average reward of the best arm, k_t the arm pulled at time t.

The bandit problem

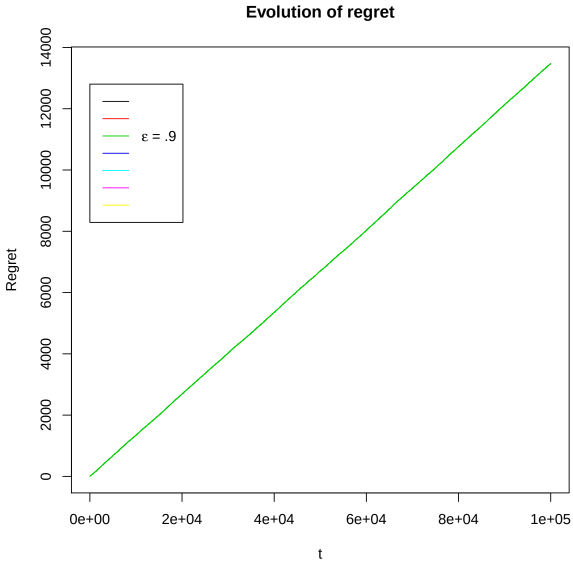
Some strategies: experimental results

Let's look at some preliminary and elementary experimental results.

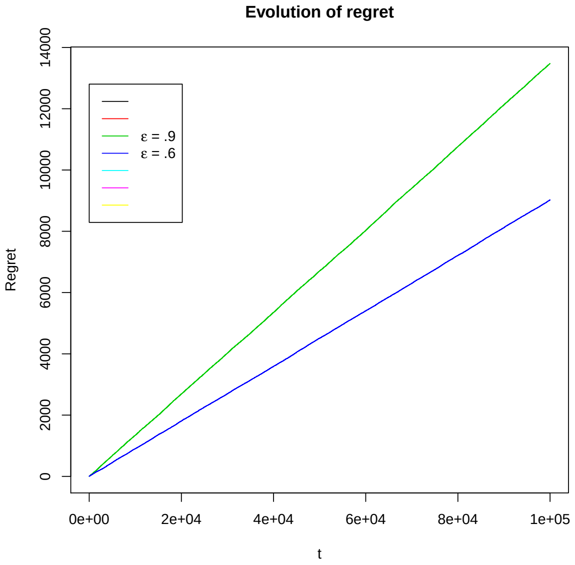
We compare various strategies:

- ϵ-greedy with $\epsilon \in \{0.9, 0.6, 0.2\}$
- ϵ-decreasing greedy with $\epsilon_0 = 0.9, \epsilon_t = \frac{1}{\sqrt{t}}$
- softmax with $\tau_0 = 5$ and $\tau_t \leftarrow 0.999\tau_{t-1}$.

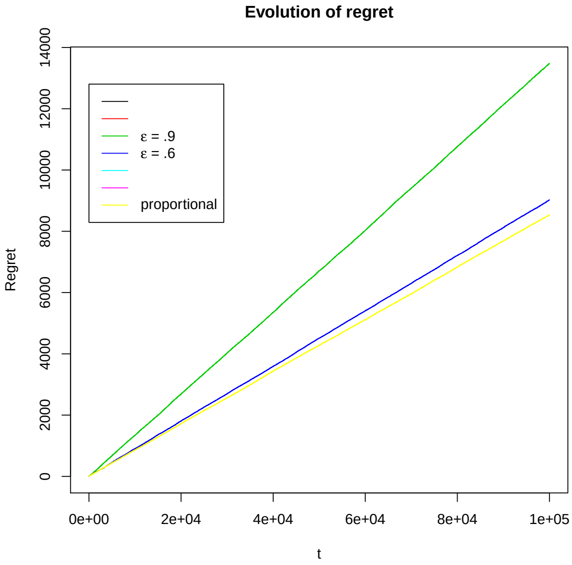
Experimental results



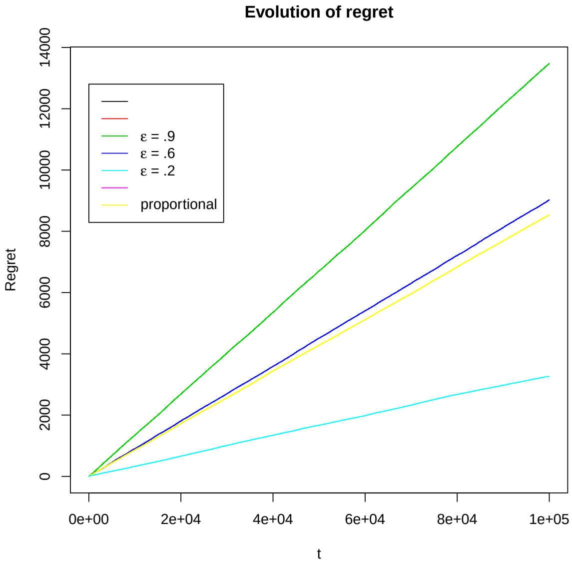
Experimental results



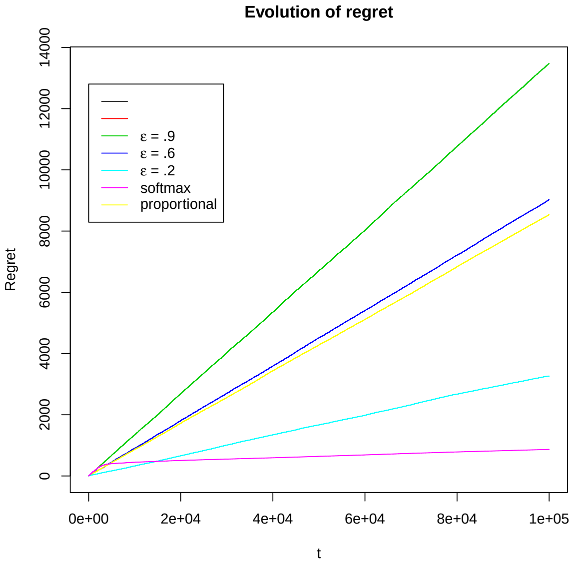
Experimental results



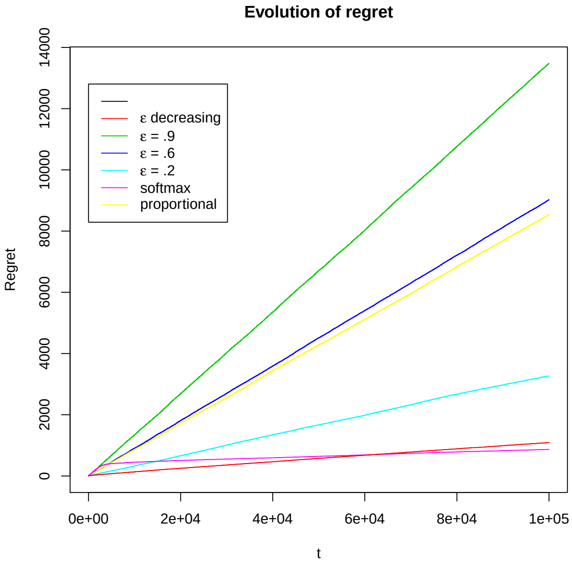
Experimental results



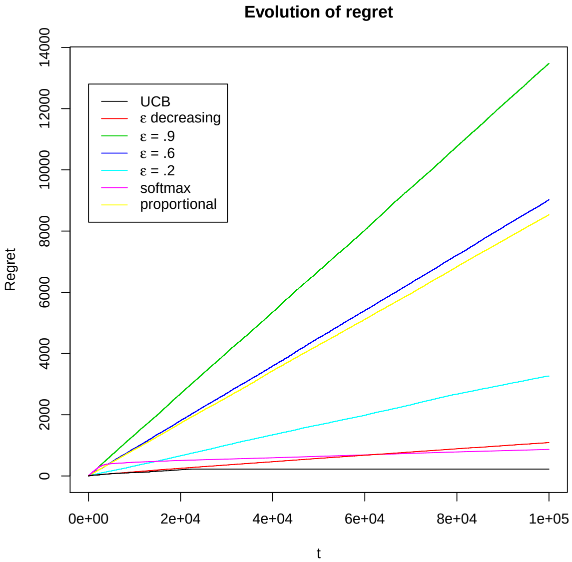
Experimental results



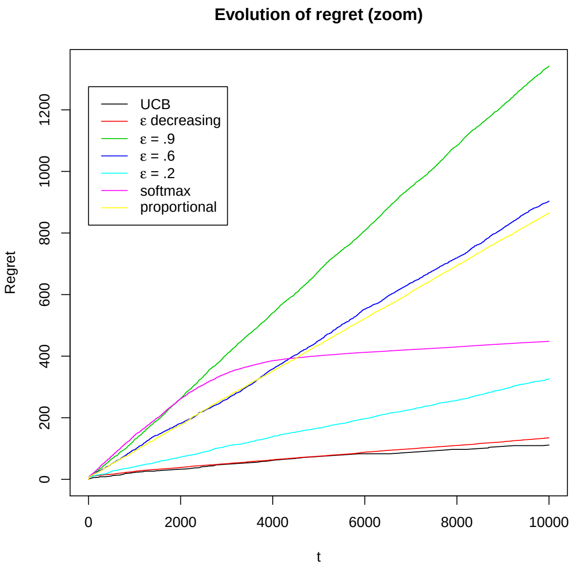
Experimental results



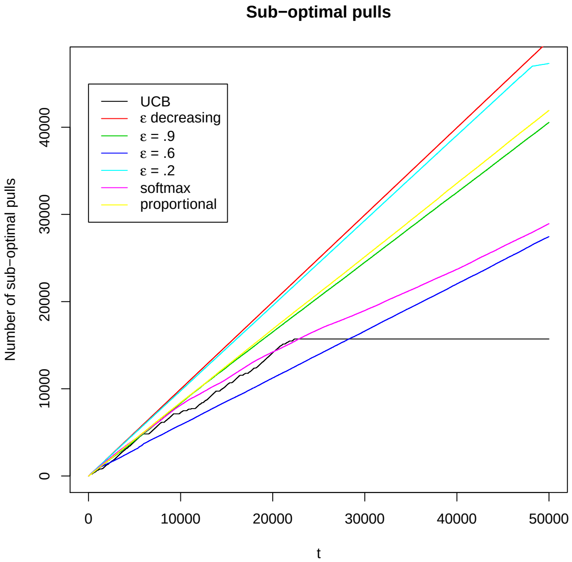
Experimental results



Experimental results



Experimental results



- UCB selects the arm that seems to be the most rewarding and the most informative:

The bandit problem
UCB

- UCB selects the arm that seems to be the most rewarding and the most informative:
  - short term benefit: greedy choice: exploit

The bandit problem
UCB

- UCB selects the arm that seems to be the most rewarding and the most informative:
  - short term benefit: greedy choice: exploit
  - long term benefit: in case of doubt, decrease the uncertainty: explore

The bandit problem
UCB

- UCB selects the arm that seems to be the most rewarding and the most informative:
  - short term benefit: greedy choice: exploit
  - long term benefit: in case of doubt, decrease the uncertainty: explore
- solves the exploitation vs. exploration dilemma

The bandit problem
UCB

- UCB selects the arm that seems to be the most rewarding and the most informative:
  - short term benefit: greedy choice: exploit
  - long term benefit: in case of doubt, decrease the uncertainty: explore
- solves the exploitation vs. exploration dilemma
- UCB pulls arm: $\arg\max_k \hat{\mu}_k + \sqrt{2\frac{\log t}{n_k}}$

The bandit problem

Some strategies

$K = 4$ arms, $t = 26$ pulls:

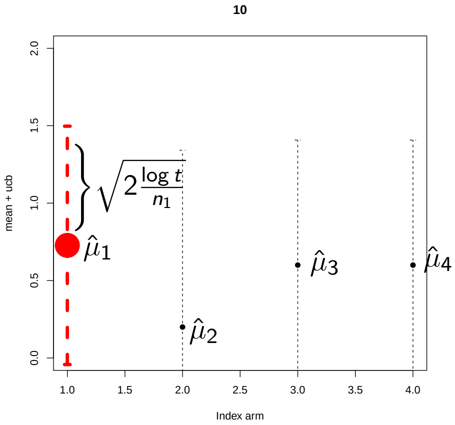



$\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6 \quad \hat{\mu}_2 = \frac{1}{4} = .25 \quad \hat{\mu}_3 = \frac{6}{10} = .6 \quad \hat{\mu}_4 = \frac{5}{7} = .71$

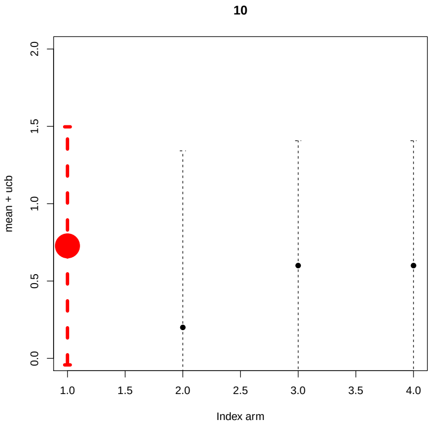
Which arm does UCB pull next?

- UCB pulls arm: $\arg\max_k \hat{\mu}_k + \sqrt{2\frac{\log t}{n_k}}$
- $\hat{\mu}_1 = \frac{3 \text{ successes}}{n_1 = 5 \text{ pulls}} = .6$, $\hat{\mu}_2 = \frac{1}{4} = .25$, $\hat{\mu}_3 = \frac{6}{10} = .6$, $\hat{\mu}_4 = \frac{5}{7} = .71$
- $\frac{3}{5} + \sqrt{2\frac{\log 26}{5}}$, $\frac{1}{4} + \sqrt{2\frac{\log 26}{4}}$, $\frac{6}{10} + \sqrt{2\frac{\log 26}{10}}$, $\frac{5}{7} + \sqrt{2\frac{\log 26}{7}}$
- $1.74, 1.53, 1.41, 1.68$
- $\rightsquigarrow$ UCB pulls arm 1

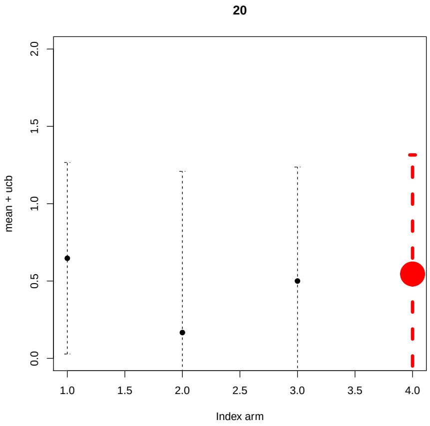
The bandit problem
UCB



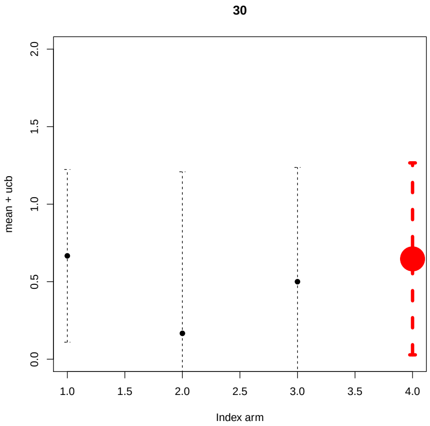
The bandit problem
UCB



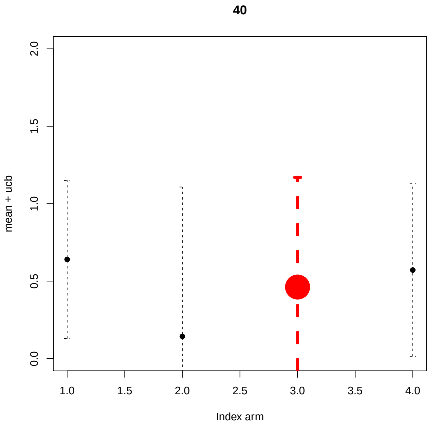
The bandit problem
UCB



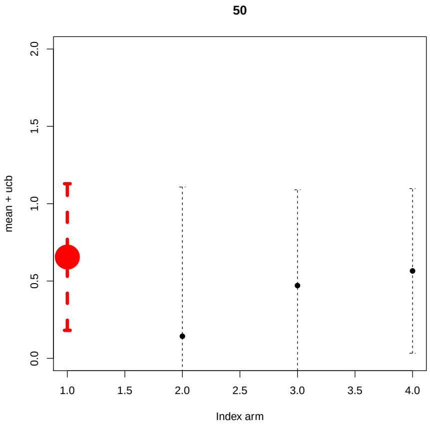
The bandit problem
UCB



The bandit problem
UCB

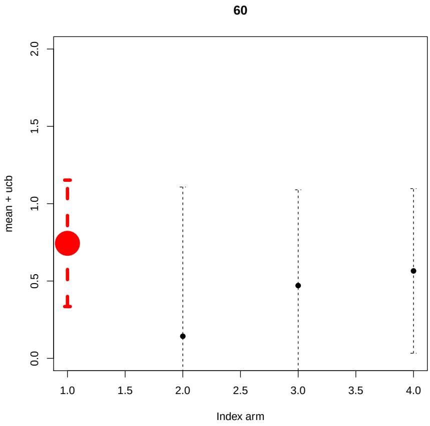


The bandit problem
UCB

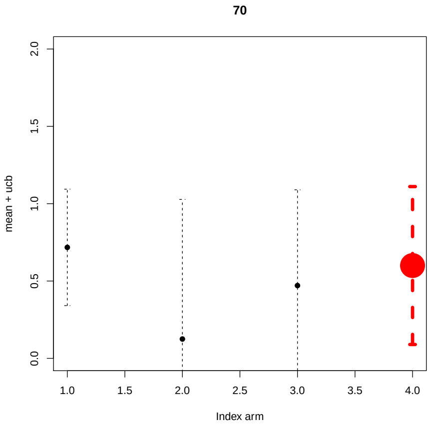


The bandit problem

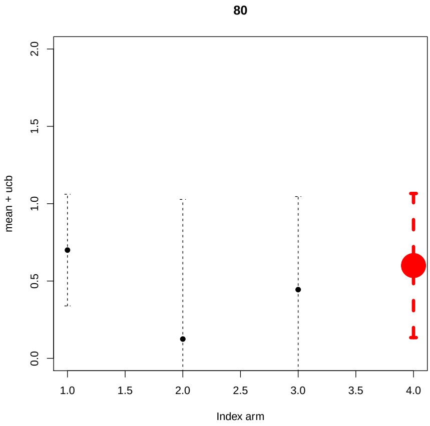
UCB



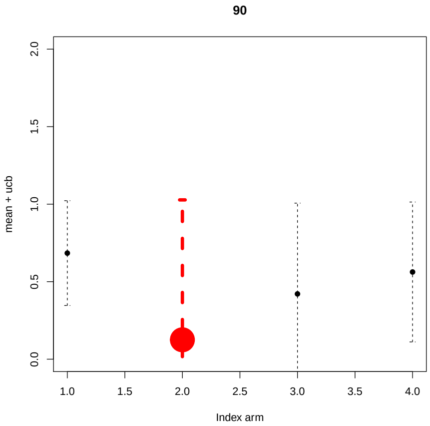
The bandit problem
UCB



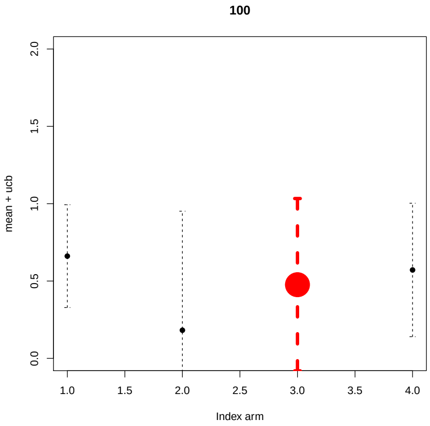
The bandit problem
UCB



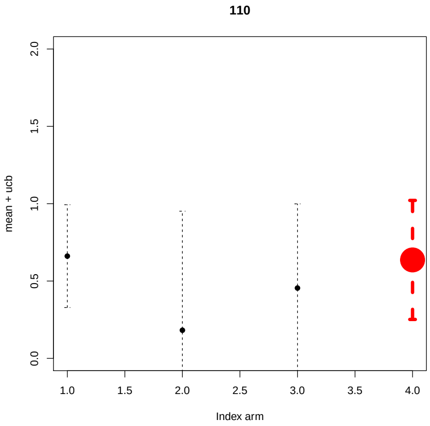
The bandit problem
UCB



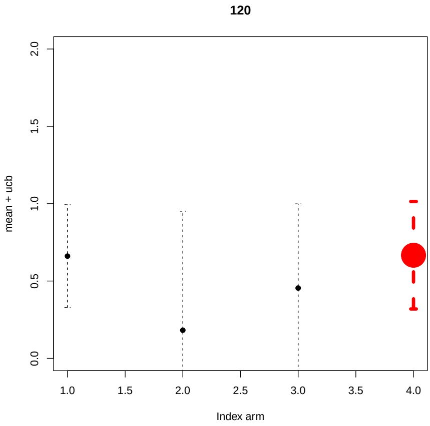
The bandit problem
UCB



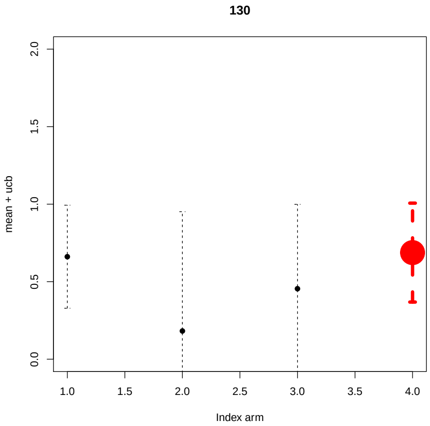
The bandit problem
UCB



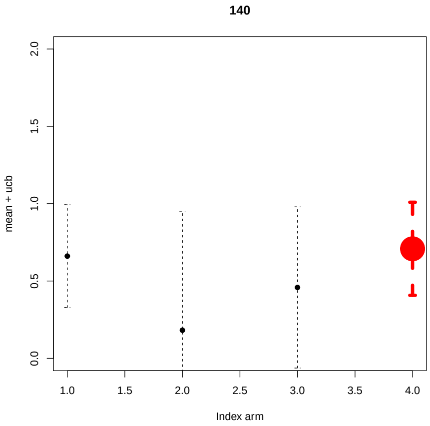
The bandit problem
UCB



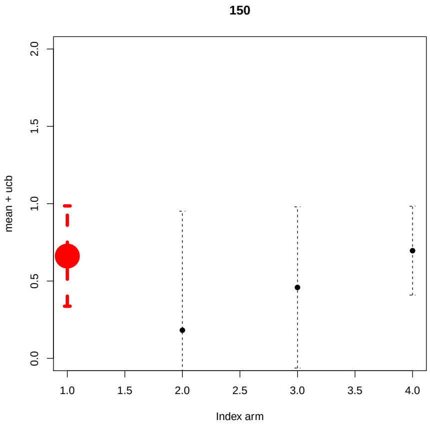
The bandit problem
UCB



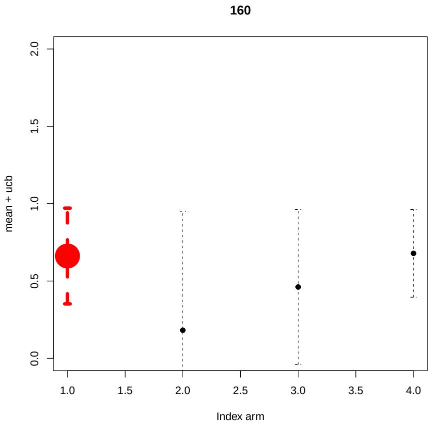
The bandit problem
UCB



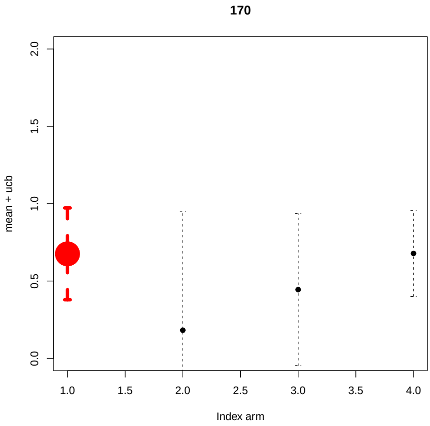
The bandit problem
UCB



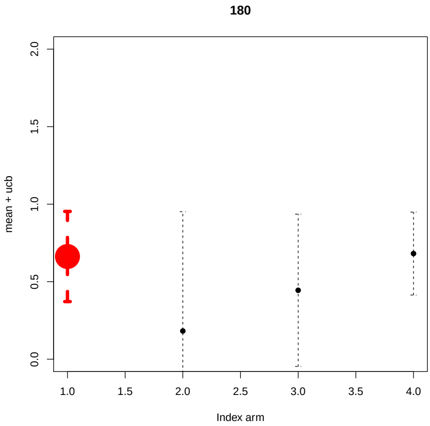
The bandit problem
UCB



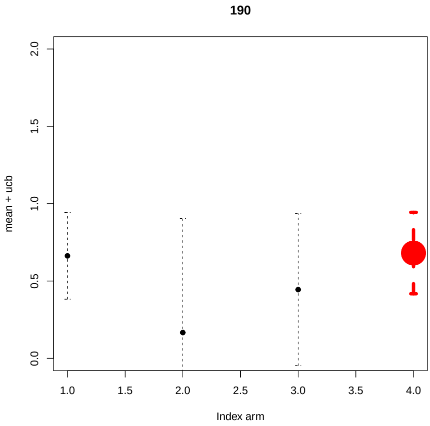
The bandit problem
UCB



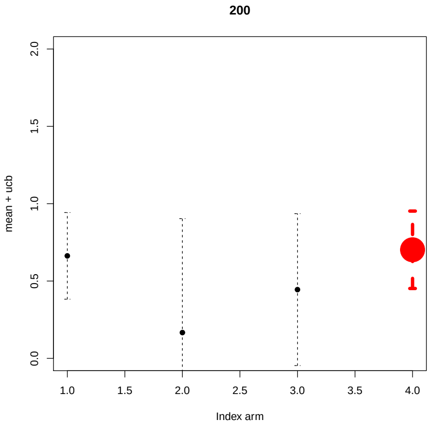
The bandit problem
UCB



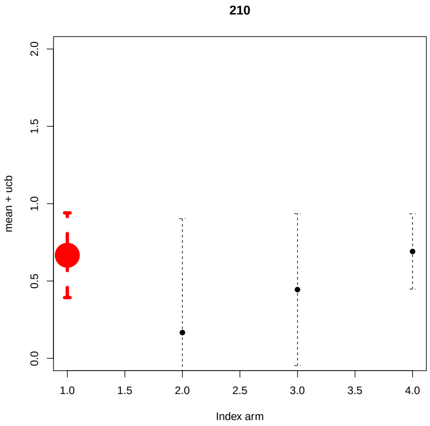
The bandit problem
UCB



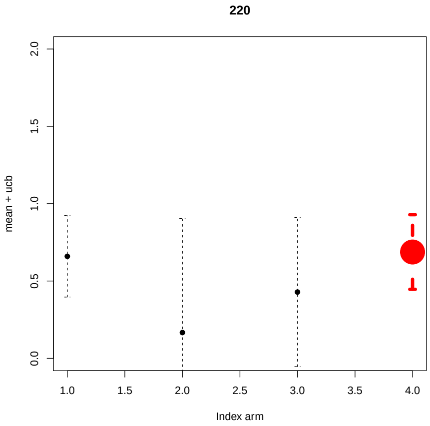
The bandit problem
UCB



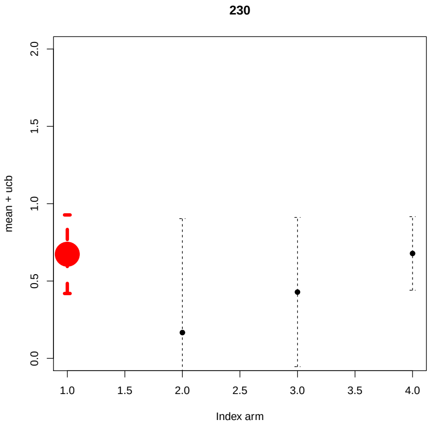
The bandit problem
UCB



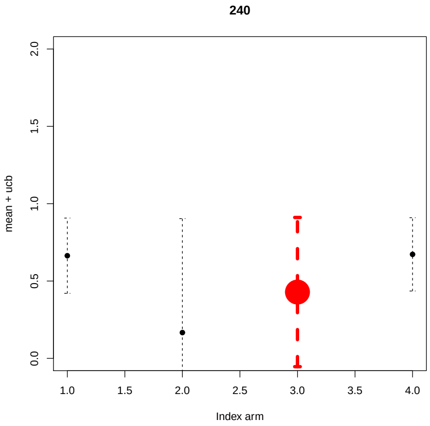
The bandit problem
UCB



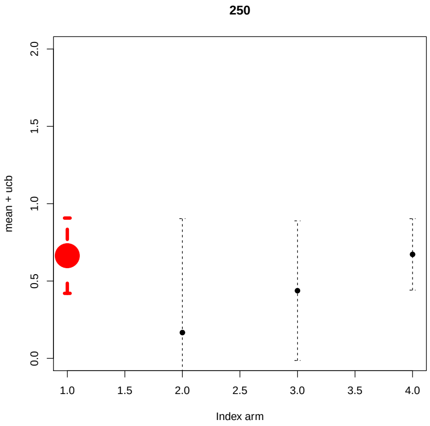
The bandit problem
UCB



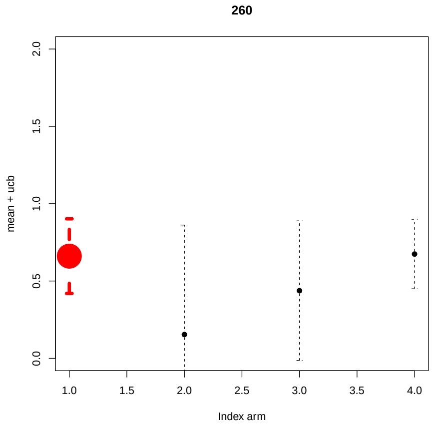
The bandit problem
UCB



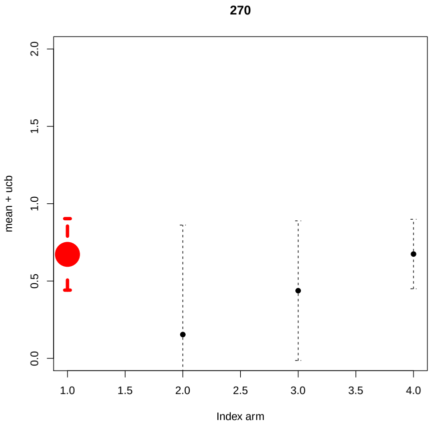
The bandit problem
UCB



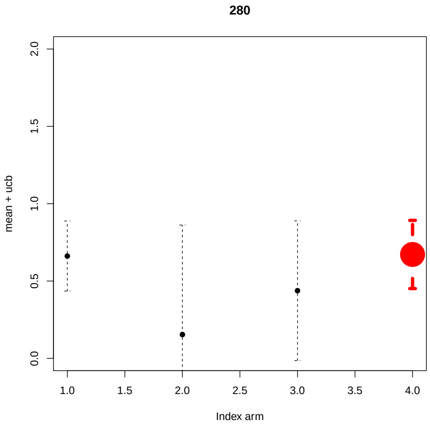
The bandit problem
UCB



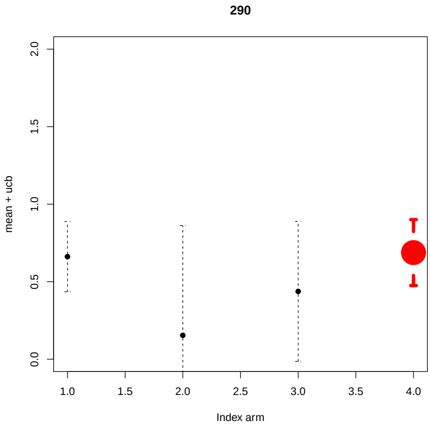
The bandit problem
UCB



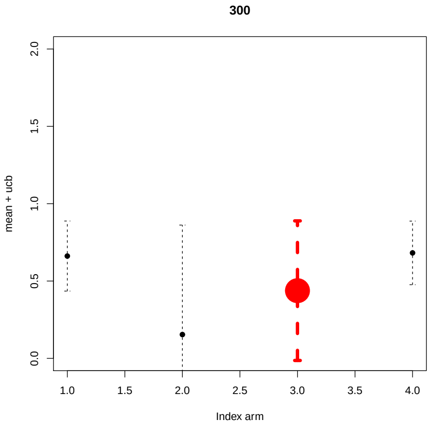
The bandit problem
UCB



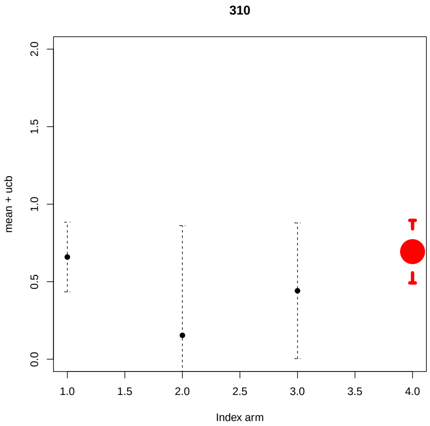
The bandit problem
UCB



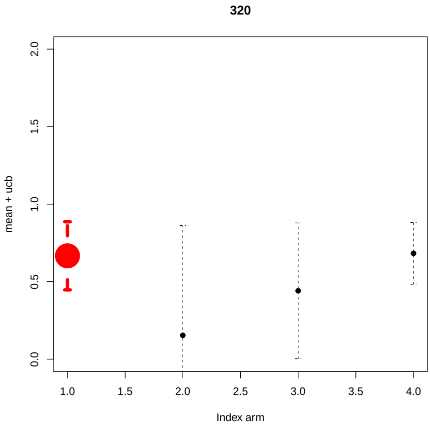
The bandit problem
UCB



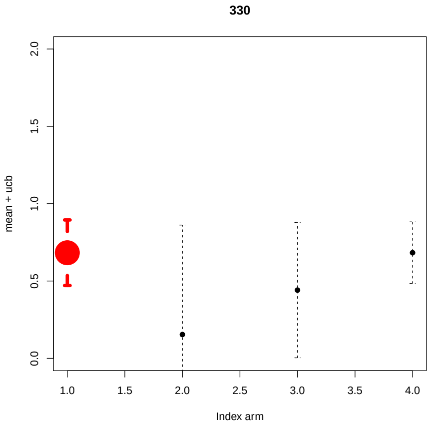
The bandit problem
UCB



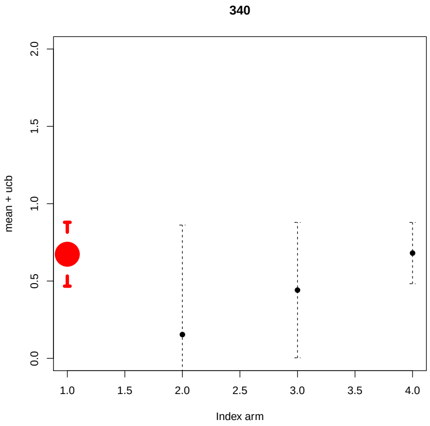
The bandit problem
UCB



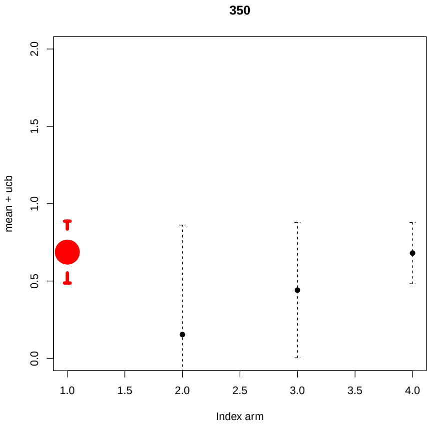
The bandit problem
UCB



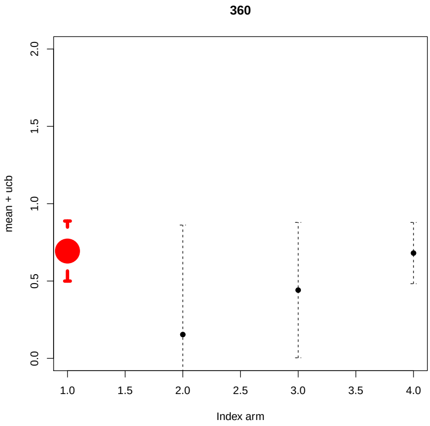
The bandit problem
UCB



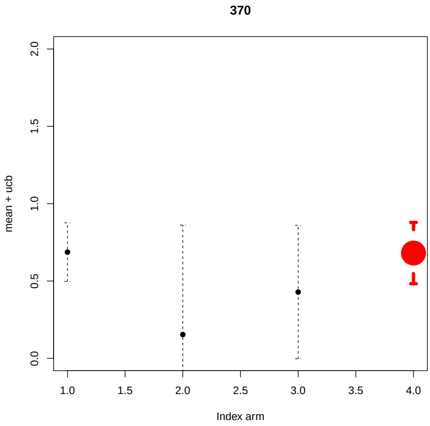
The bandit problem
UCB



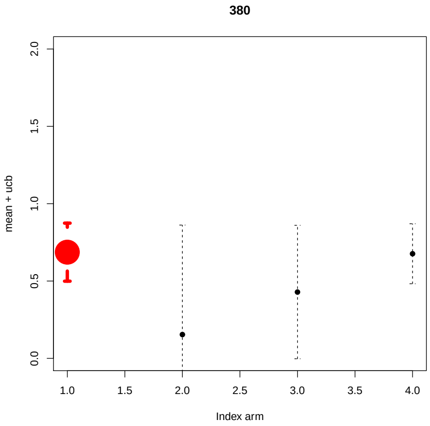
The bandit problem
UCB



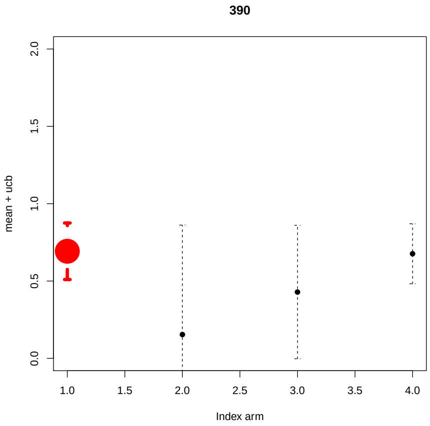
The bandit problem
UCB



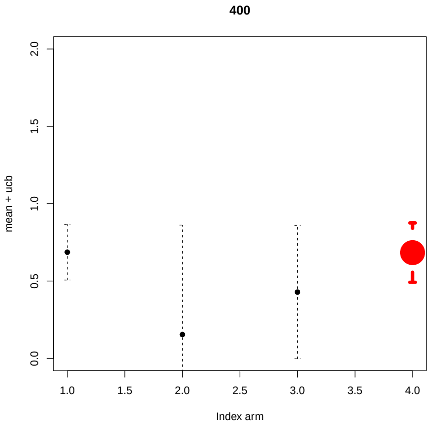
The bandit problem
UCB



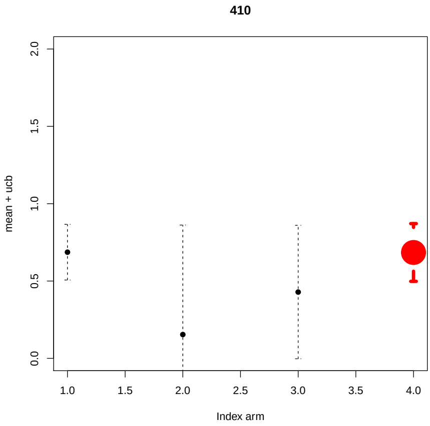
The bandit problem
UCB



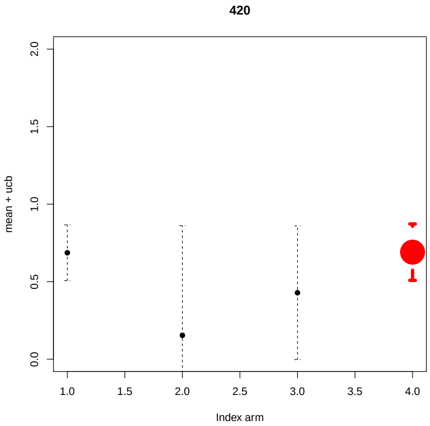
The bandit problem
UCB



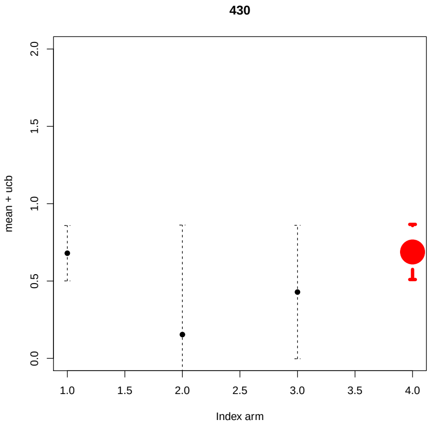
The bandit problem
UCB



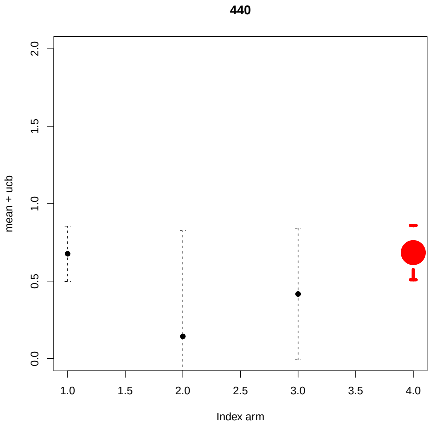
The bandit problem
UCB



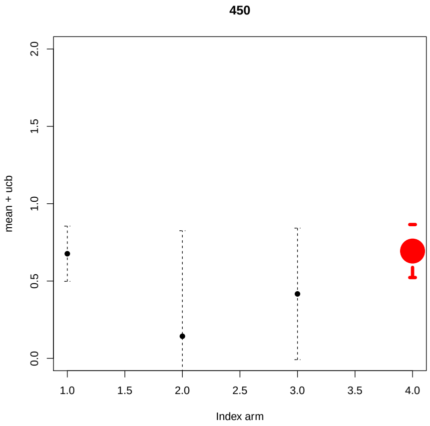
The bandit problem
UCB



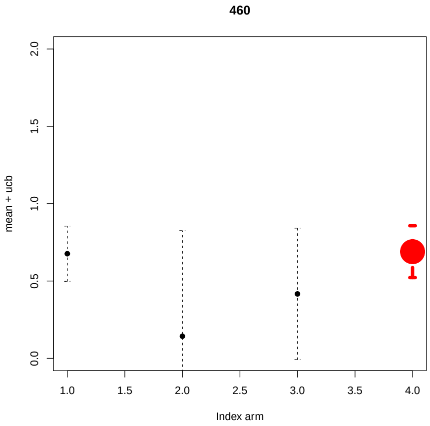
The bandit problem
UCB



The bandit problem
UCB

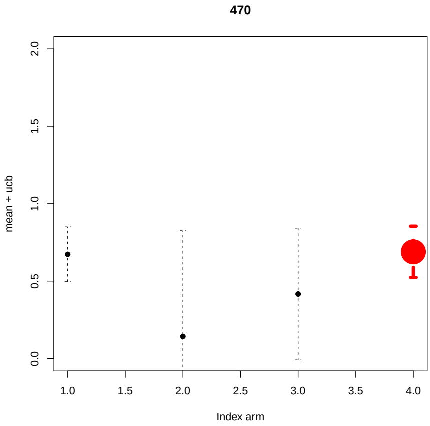


The bandit problem
UCB

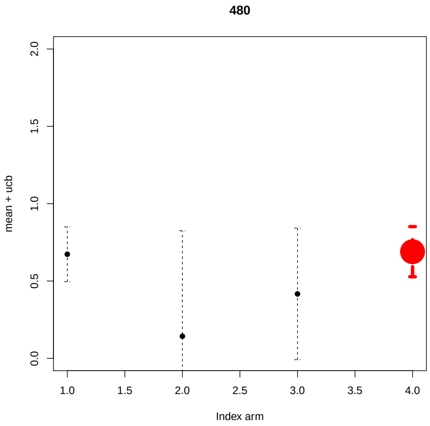


The bandit problem

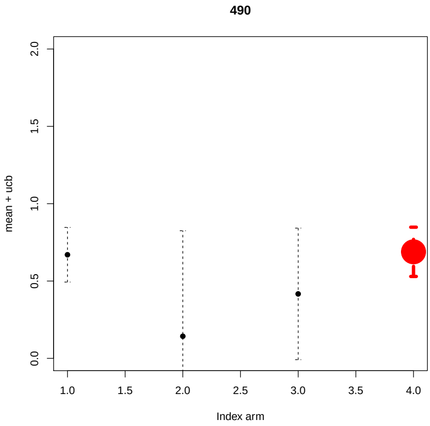
UCB



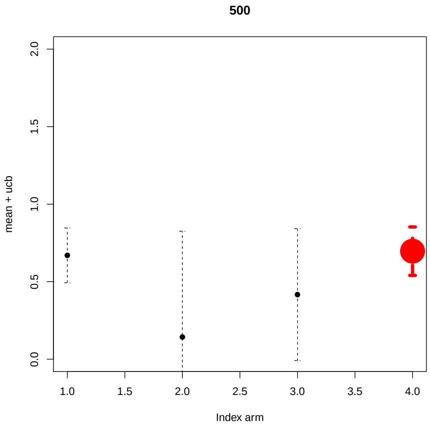
The bandit problem
UCB



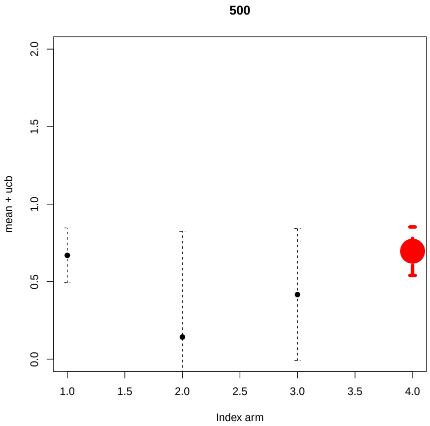
The bandit problem
UCB



The bandit problem
UCB



The bandit problem
UCB



The 4th arm is indeed the best.

The bandit problem

The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

  That's the best achievable.

The bandit problem
The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

  That's the best achievable.

- This is optimal up to some constants.

The bandit problem

The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

  That's the best achievable.
- This is optimal up to some constants.
- Why this bound? $(\sqrt{2\frac{\log t}{n_k}})$

The bandit problem

The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

  That's the best achievable.
- This is optimal up to some constants.
- Why this bound? $(\sqrt{2\frac{\log t}{n_k}})$
- Answer: to be able to proove a logarithmic mean performance.

  Based on Hoeffding's inequality

The bandit problem

The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

  That's the best achievable.

- This is optimal up to some constants.
- Why this bound? $(\sqrt{2\frac{\log t}{n_k}})$
- Answer: to be able to proove a logarithmic mean performance.

  Based on Hoeffding's inequality

- The 2 is a constant that makes the proof work.

The bandit problem

The optimistic approach

- UCB enjoys a $\mathcal{O}(\log t)$ mean regret and a $\mathcal{O}(\sqrt{Kt})$ worst case regret.

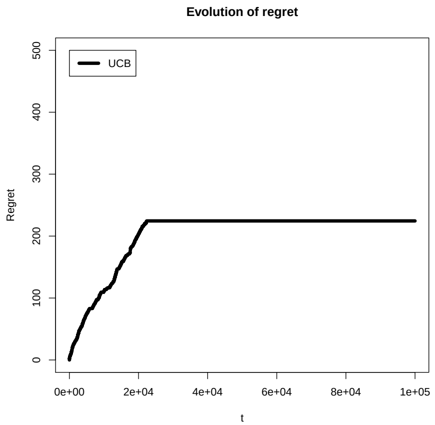
  That's the best achievable.
- This is optimal up to some constants.
- Why this bound? ($\sqrt{2\frac{\log t}{n_k}}$)
- Answer: to be able to proove a logarithmic mean performance.

  Based on Hoeffding's inequality
- The 2 is a constant that makes the proof work.
- In practice, this is a parameter to tune.

The bandit problem

The UCB family

$$\arg\max_k \hat{\mu}_k + \sqrt{\frac{\alpha \log t}{n_k}}$$

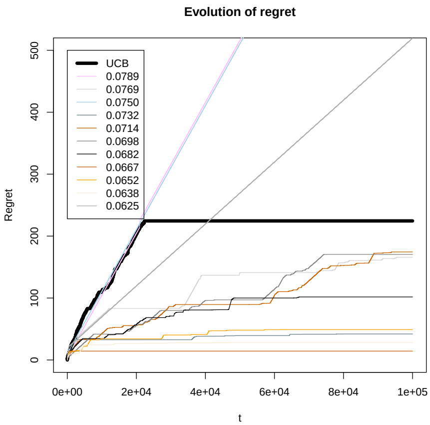


The bandit problem

The UCB family

$$\arg\max_k \hat{\mu}_k + \sqrt{\frac{\alpha \log t}{n_k}}$$

Tuning α:



The bandit problem
The UCB family

UCB variants:

- using the variance (UCB-V)
- based on the distribution of returns (KL-UCB, BESA)
- ...

The bandit problem
The UCB family

UCB variants:

- using the variance (UCB-V)
- based on the distribution of returns (KL-UCB, BESA)
- ...

The growing set of bandit problems:

- structured bandits:
  - contextual bandits (bandits with side information): linUCB, kernelUCB
  - combinatorial bandits
  - ...
- bandits with costs (arm pulling, changing arm, ...)
- mortal bandits
- ...

The bandit problem
The UCB family

UCB variants:

- using the variance (UCB-V)
- based on the distribution of returns (KL-UCB, BESA)
- ...

The growing set of bandit problems:

- structured bandits:
  - contextual bandits (bandits with side information): linUCB, kernelUCB
  - combinatorial bandits
  - ...
- bandits with costs (arm pulling, changing arm, ...)
- mortal bandits
- ...

Other regrets:

- pure regret: only the regret at the last turn matters
- risk-aware regret: avoid bad actions
- ...

The bandit problem
The UCB family

UCB variants:

- using the variance (UCB-V)
- based on the distribution of returns (KL-UCB, BESA)
- ...

The growing set of bandit problems:

- structured bandits:
  - contextual bandits (bandits with side information): linUCB, kernelUCB
  - combinatorial bandits
  - ...
- bandits with costs (arm pulling, changing arm, ...)
- mortal bandits
- ...

Other regrets:

- pure regret: only the regret at the last turn matters
- risk-aware regret: avoid bad actions
- ...

and other families: Thompson sampling, Gittins indices.

The bandit problem
The UCB family

UCB variants:

- using the variance (UCB-V)
- based on the distribution of returns (KL-UCB, BESA)
- ...

The growing set of bandit problems:

- **structured** bandits:
  - contextual bandits (bandits with side information): linUCB, kernelUCB
  - combinatorial bandits
  - ...
- bandits with costs (arm pulling, changing arm, ...)
- mortal bandits
- ...

Other regrets:

- pure regret: only the regret at the last turn matters
- risk-aware regret: avoid bad actions
- ...

and other families: Thompson sampling, Gittins indices.

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits, a structure.

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits, a structure.
- **Key idea**: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.

The bandit problem

Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits,
  a structure.
- Key idea: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.
- How does this "side" information impact regret bounds? (algorithm complexity)?
  Can it reduce the complexity significantly?

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits, a structure.
- Key idea: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.
- How does this "side" information impact regret bounds? (algorithm complexity)?
  Can it reduce the complexity significantly?
- **Yes, very much, in theory and in pratice.**

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits,
  a structure.
- Key idea: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.
- How does this “side” information impact regret bounds? (algorithm complexity)?
  Can it reduce the complexity significantly?
- Yes, very much, in theory and in pratice.
- How can we exploit it in algorithms?

The bandit problem
Structured bandits

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits, a structure.
- Key idea: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.
- How does this "side" information impact regret bounds? (algorithm complexity)?
  Can it reduce the complexity significantly?
- Yes, very much, in theory and in pratice.
- How can we exploit it in algorithms?
- How can we exploit it in real applications?

- How to deal with many bandits: K large, or even $K = \infty$?
- In realistic settings, there are relations between bandits, a structure.
- Key idea: when an arm is pulled, you obtain information about this pulled arm and other, related, arms.
- How does this "side" information impact regret bounds? (algorithm complexity)?
  Can it reduce the complexity significantly?
- Yes, very much, in theory and in pratice.
- How can we exploit it in algorithms?
- How can we exploit it in real applications?
- How can we quantify this structure?

The bandit problem
The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$

The bandit problem
The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown

The bandit problem
The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown
- Linear bandit: linUCB
  at each t, pull arm $\arg\max_k \langle w, \phi_k \rangle + \alpha \sqrt{\phi_k \mathbf{A}^{-1} \phi_k^T}$
  where $\mathbf{A} = \sum_t \phi_{k_t} \phi_{k_t}^T + \mathbf{Id}$

The bandit problem

The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown
- Linear bandit: linUCB
  at each t, pull arm $\arg\max_k \langle w, \phi_k \rangle + \alpha \sqrt{\phi_k \mathbf{A}^{-1} \phi_k^T}$
  where $\mathbf{A} = \sum_t \phi_{k_t} \phi_{k_t}^T + \mathbf{Id}$
- Worst case regret: $\mathcal{O}(\sqrt{Pt})$ instead of $\mathcal{O}(\sqrt{Kt})$

The bandit problem
The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown
- Linear bandit: linUCB
  at each t, pull arm $\arg\max_k \langle w, \phi_k \rangle + \alpha \sqrt{\phi_k \mathbf{A}^{-1} \phi_k^T}$
  where $\mathbf{A} = \sum_t \phi_{k_t} \phi_{k_t}^T + \mathbf{Id}$
- Worst case regret: $\mathcal{O}(\sqrt{Pt})$ instead of $\mathcal{O}(\sqrt{Kt})$
- Example application: recommendation systems: $K \approx 10^6$ items;
  $P \approx 100 \rightsquigarrow$ regret $/100$.

The bandit problem

The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown
- Linear bandit: linUCB
  at each t, pull arm $\arg\max_k \langle w, \phi_k \rangle + \alpha \sqrt{\phi_k \mathbf{A}^{-1} \phi_k^T}$
  where $\mathbf{A} = \sum_t \phi_{k_t} \phi_{k_t}^T + \mathbf{Id}$
- Worst case regret: $\mathcal{O}(\sqrt{Pt})$ instead of $\mathcal{O}(\sqrt{Kt})$
- Example application: recommendation systems: $K \approx 10^6$ items; $P \approx 100 \rightsquigarrow$ regret $/100$.
- Kernel trick: $\langle ., . \rangle \to k(., .)$: linUCB $\to$ kernelUCB

The bandit problem
The linear UCB

- Assume each bandit k has side information, a set of features $\phi_k \in \mathbb{R}^P$
- Assume $\mu_k = \langle w, \phi_k \rangle$, w unknown
- Linear bandit: linUCB
  at each t, pull arm $\arg\max_k \langle w, \phi_k \rangle + \alpha \sqrt{\phi_k \mathbf{A}^{-1} \phi_k^T}$
  where $\mathbf{A} = \sum_t \phi_{k_t} \phi_{k_t}^T + \mathbf{Id}$
- Worst case regret: $\mathcal{O}(\sqrt{Pt})$ instead of $\mathcal{O}(\sqrt{Kt})$
- Example application: recommendation systems: $K \approx 10^6$ items; $P \approx 100 \rightsquigarrow$ regret $/100$.
- Kernel trick: $\langle ., . \rangle \rightarrow k(., .)$: linUCB $\rightarrow$ kernelUCB
- Algorithmic complexity scales with the complexity of the problem (P) rather than its size (K).

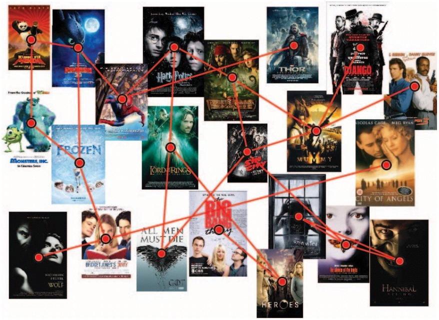
The bandit problem

Graphs of bandits

- In the context of a RecSys, let us consider a graph in which:
  - 1 vertex = 1 product = 1 bandit
  - pulling an arm $\rightsquigarrow$ rating of a product by a user/client
  - 1 edge

    = the two linked products have close average ratings

    = some relation between the two linked bandits



- reward = the happiness of the user (whichever meaning you put in it!)

The bandit problem

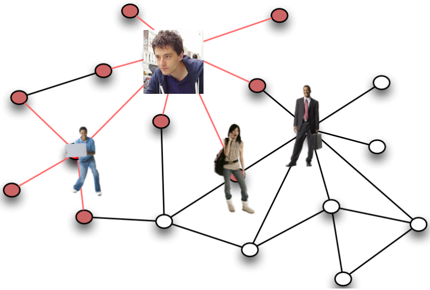
Graphs of bandits

- We can assume reward smoothness along edges
- Task: find the best products (those with highest average rate): block busters, ...
- spectralUCB: regret scales with the effective dimension D_{eff} of the graph, *i.e.* the number of relevant eigenvectors of the graph Laplacian $\ll$ number of vertices
- Algorithmic complexity scales with the complexity of the problem (D_{eff}) rather than its size (K).

The bandit problem

Graphs of bandits

- Let us consider a social network size graph.
- Who are the most influential people?
- only local knowledge of the graph: nodes connected to a given node (full graph is unknown and continuously evolving)



The bandit problem

Applications

- computational advertizing
- recommendation systems
- web content personnalization

Methodology

A change of perspective

Turning a problem into a sequential decision making problem.

Bandits for recommendation systems

Main approaches:

- content-based
  based on product description/features; side/contextual information
  $\rightsquigarrow$ nearest neighbors of some sort
- collaborative filtering
  based on user ratings: a user is described by its ratings; likewise for products
  $\rightsquigarrow$ matrix factorization
- hybrid

Bandits for recommendation systems

Turning collab. filtering to a hybrid approach

- side information of products? of users?

Bandits for recommendation systems

Turning collab. filtering to a hybrid approach

- side information of products? of users?
- Idea:
  represent products by user satisfaction, and
  represent users with product they like/dislike

Bandits for recommendation systems

Turning collab. filtering to a hybrid approach

- side information of products? of users?
- Idea:
  represent products by user satisfaction, and
  represent users with product they like/dislike
- **Goal**: Learn side information to make good recommendations.

- side information of products? of users?
- Idea:
  represent products by user satisfaction, and
  represent users with product they like/dislike
- **Goal**: Learn side information to make good recommendations.
- **Basic information**: ratings (t, u, p, r)
  t: time; u: user; p: product; r: rating.

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\rightsquigarrow$ low rank of $\mathbf{R}$ assumption

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\leadsto$ low rank of $\mathbf{R}$ assumption
- $\leadsto$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^\mathbf{T}$

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\rightsquigarrow$ low rank of $\mathbf{R}$ assumption
- $\rightsquigarrow$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathbf{T}}$
- $\mathbf{U}_i$ represents user i in a latent space

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\rightsquigarrow$ low rank of $\mathbf{R}$ assumption
- $\rightsquigarrow$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathbf{T}}$
- $\mathbf{U}_i$ represents user i in a latent space
- likewise for $\mathbf{V}_j$ for products

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\rightsquigarrow$ low rank of $\mathbf{R}$ assumption
- $\rightsquigarrow$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathbf{T}}$
- $\mathbf{U}_i$ represents user i in a latent space
- likewise for $\mathbf{V}_j$ for products

- Use these features in linUCB to select items to recommend

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\leadsto$ low rank of $\mathbf{R}$ assumption
- $\leadsto$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathbf{T}}$
- $\mathbf{U}_i$ represents user i in a latent space
- likewise for $\mathbf{V}_j$ for products

- Use these features in linUCB to select items to recommend
- These features are latent factors

Bandits for recommendation systems

Matrix factorization provides latent features

- Let $\mathbf{R}$ be the $N \times P$ user/product rating matrix.
- $r_{i,j}$ is the rating of user i on product j
- most (99.9%) of them are unknown
- users and products are clustered
- $\rightsquigarrow$ low rank of $\mathbf{R}$ assumption
- $\rightsquigarrow$ find $\mathbf{U} \in \mathbb{R}^{N \times K}$ and $\mathbf{V} \in \mathbb{R}^{P \times K}$ such that $\mathbf{R} = \mathbf{U}\mathbf{V}^{\mathbf{T}}$
- $\mathbf{U}_i$ represents user i in a latent space
- likewise for $\mathbf{V}_j$ for products

- Use these features in linUCB to select items to recommend
- These features are latent factors
- May be mixed with other available attributes

Bandits for function optimization

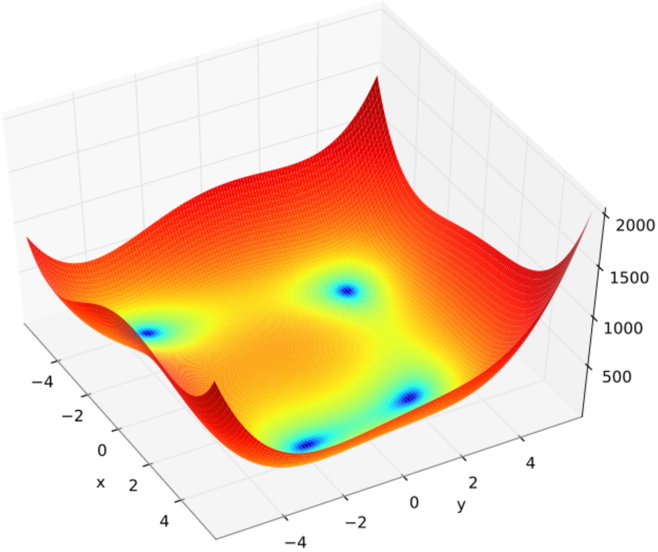
Recommendation and tutoring

- recommend exercises, lectures, … that are the most likely to be useful to a trainee
- here, bandits are:
  - contextual,
  - in a time varying environment: the trainee learns.
- bandits look for an effective and efficient training:
  - useless to provide too easy exercises, useless to provide repeatedly the same exercise.
  - Diversity of types of exercise, topics of exercises.
  - Keep track and select arms according to the learning curve of the trainee.
  - Personal experience: the worst that can happen: a student bores. Not because it is too difficult for her, but because it is too easy.

Live on `https://www.afterclasse.fr/`

Partially funded and in partnership with Le Livre Scolaire.

Bandits for function optimization



Bandits for function optimization

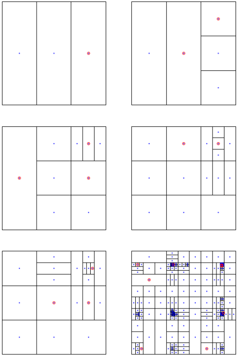
- f, a **noisy** function:
  $f = f^* + \eta : \mathbb{R}^P \to \mathbb{R}$
- find a **global** minimum:
  $x^*, \arg\min(f^*)$
- f unknown.

Bandits for function optimization

- f, a **noisy** function:
  $f = f^* + \eta : \mathbb{R}^P \rightarrow \mathbb{R}$
- find a **global** minimum:
  $x^*, \arg\min(f^*)$
- f unknown.
- assume each point $x\mathbb{R}^P$ is a bandit arm with expected loss $f^*(x)$
- assume an unknown regularity around the optimum (smooth, Lipschitz, ...)
- SOO

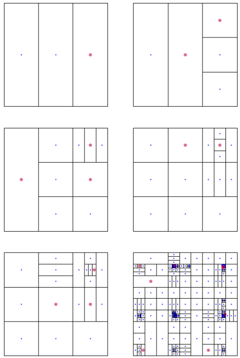
Bandits for function optimization

- f, a **noisy** function:
  $f = f^* + \eta : \mathbb{R}^P \rightarrow \mathbb{R}$
- find a **global** minimum:
  $x^*, \arg\min(f^*)$
- f unknown.
- assume each point $x\mathbb{R}^P$ is a bandit arm with expected loss $f^*(x)$
- assume an unknown regularity around the optimum (smooth, Lipschitz, ...)
- SOO



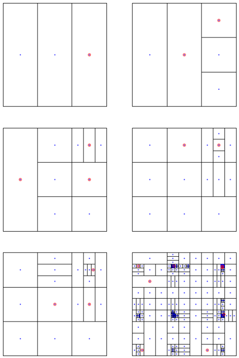
Bandits for function optimization

- f, a **noisy** function:
  $f = f^* + \eta : \mathbb{R}^P \to \mathbb{R}$
- find a **global** minimum:
  $x^*, \arg\min(f^*)$
- f unknown.
- assume each point $x\mathbb{R}^P$ is a bandit arm with expected loss $f^*(x)$
- assume an unknown regularity around the optimum (smooth, Lipschitz, ...)
- SOO converges asymptotically to a global minimum
- its regret scales as $\mathcal{O}(t^{-\frac{1}{d}})$



Bandits for function optimization

- f, a **noisy** function: $f = f^* + \eta : \mathbb{R}^P \rightarrow \mathbb{R}$
- find a **global** minimum: $x^*, \arg\min(f^*)$
- f unknown.
- assume each point $x\mathbb{R}^P$ is a bandit arm with expected loss $f^*(x)$
- assume an unknown regularity around the optimum (smooth, Lipschitz, ...)
- SOO converges asymptotically to a global minimum
- its regret scales as $\mathcal{O}(t^{-\frac{1}{d}})$



- Reasonnable performance in the CEC 2014 challenge
- A whole family of algorithms.
- SOO builds a tree of bandits.

The bandit problem

Trees of bandits: the MCTS revolution

- a large tree that can not be exhaustively searched (*e.g.* game of chess, go, ...)
  Chess: branch factor ≈ 20; tree depth ≈ 40
  Go: branch factor up to 400; tree depth ≈ 400
- simulate at random games based on the current knowledge of the game
- the outcome of each simulation provides an estimate of the value of each visited leaf
- these values may be backed-up the root of the tree

- a large tree that can not be exhaustively searched (*e.g.* game of chess, go, ...)
  Chess: branch factor $\approx$ 20; tree depth $\approx$ 40
  Go: branch factor up to 400; tree depth $\approx$ 400
- simulate at random games based on the current knowledge of the game
  Selection of the branch to expand?
- the outcome of each simulation provides an estimate of the value of each visited leaf
- these values may be backed-up the root of the tree
  How?

The bandit problem

Trees of bandits: the MCTS revolution

- a large tree that can not be exhaustively searched (*e.g.* game of chess, go, ...)
  Chess: branch factor ≈ 20; tree depth ≈ 40
  Go: branch factor up to 400; tree depth ≈ 400
- simulate at random games based on the current knowledge of the game
  Selection of the branch to expand?
- the outcome of each simulation provides an estimate of the value of each visited leaf
- these values may be backed-up the root of the tree
  How?

Use bandits

Bandits in graphs

Monte Carlo Tree Search (MCTS)

- Today a main component of reinforcement learning algorithms.
- Example: in Alpha Zero:

$$\text{argmax}_a Q(s, a) + \alpha \frac{P(s,a)}{1+N(s,a)}$$

The bandit problem
Other applications

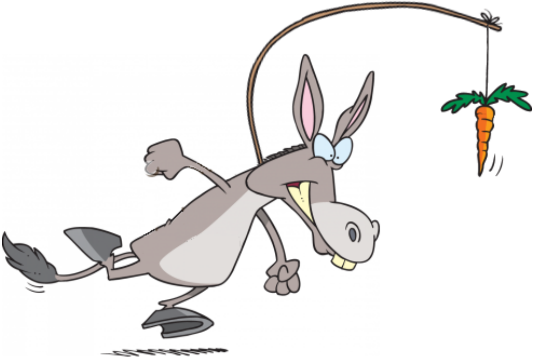
- clinical trials
- personnalized medecine/care
- cognitive radio: frequency channels allocation
- sampling in general
- smart farming
- sustainable development
- and many more

The bandit problem

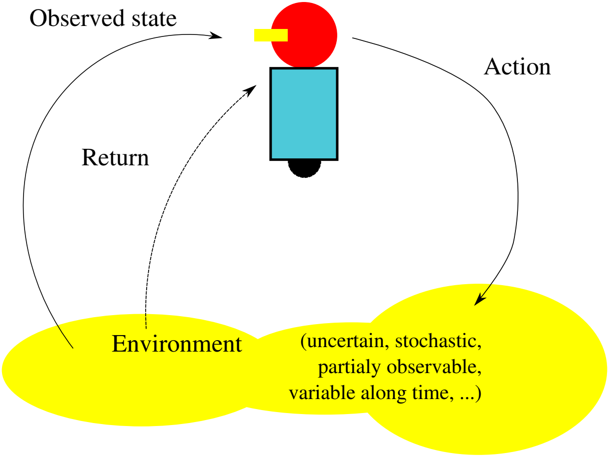
Take home message

- The bandit problem provides a framework for sampling.
- Trade-off exploration *vs.* exploitation.
- Leads to (usually) very simple algorithms.
- Amenable to formal, non asymptotic, analysis of their performance.
- Has many, and an increasing number of, applications in the real.

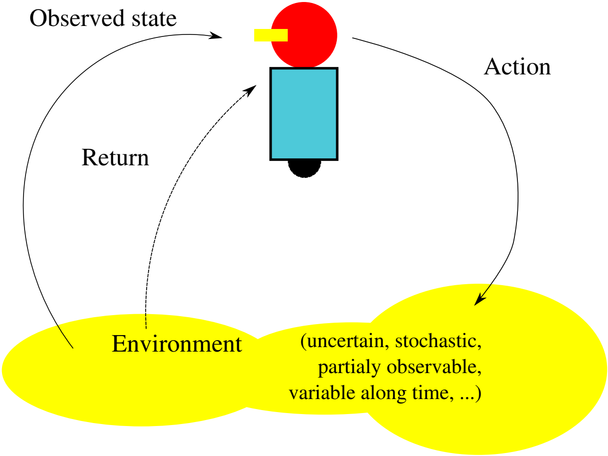
Reinforcement Learning



Reinforcement Learning



Reinforcement Learning



Learn an optimal behavior.

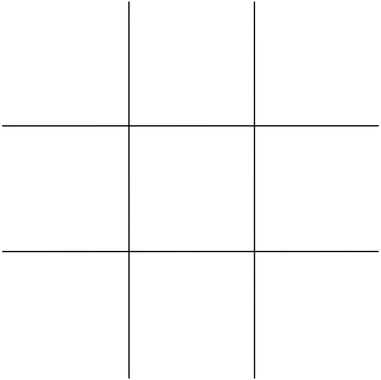
Usually:

- the environment is assumed following a Markov dynamics: the state of the environment contains all the significant information about its past, ans its knowledge is enough to make the best decision.
  observation = state
- the environment is assumed static,
- the set of states of the environment is fixed and known,
- the set of actions is fixed and known.

Going beyond these limitations is studied, and these are important issues/avenues of research.

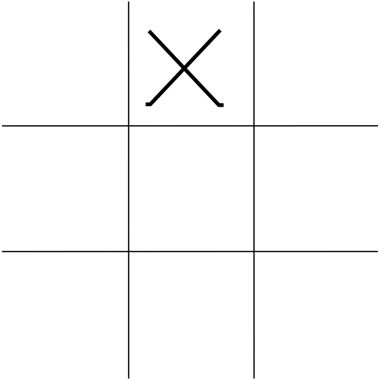
- set of instants (time) $t \in \mathcal{T}$
- set of states $x \in \mathcal{X}$
- set of actions $a \in \mathcal{A}$
- transition function: $Pr(x_{t+1}|x_t, a_t)$
- reward function: $r(x_{t+1}|x_t, a_t)$
- an objective function ζ

- **Goal**: find a policy $\pi^* : \mathcal{X} \rightarrow \mathcal{A}$ to optimize ζ
- once learned, π^* tells where to play next in order to win, or not lose, the game.



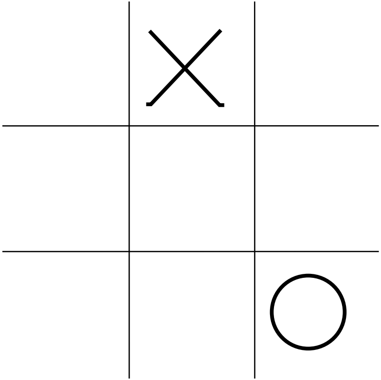
Reinforcement Learning

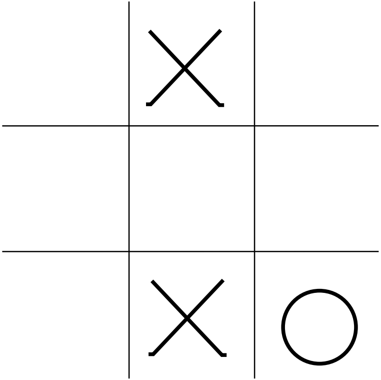
Markov Decision Problems: example on Tic-Tac-Toe



Reinforcement Learning

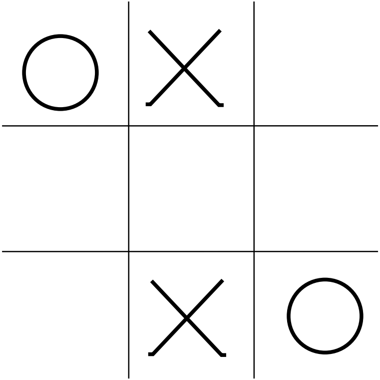
Markov Decision Problems: example on Tic-Tac-Toe

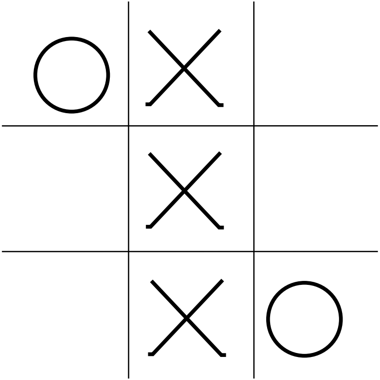


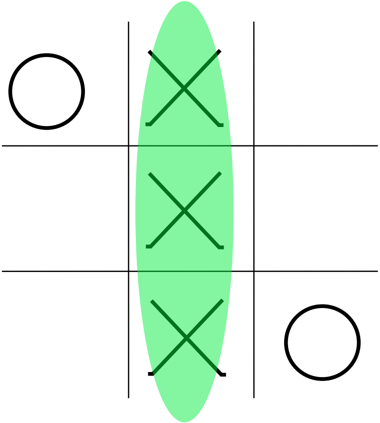


Reinforcement Learning

Markov Decision Problems: example on Tic-Tac-Toe

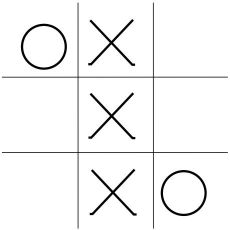






Reinforcement Learning

Markov Decision Problems: example on Tic-Tac-Toe



- set of instants (time): $0 \leq t \leq 9$
- set of states $x \in \mathcal{X}$, 3^9 of them
- set of actions $a \in \mathcal{A}$: play 1 x/o in an empty cell
- transition kernel: tic-tac-toe is deterministic
- reward function:

$$r_t = \begin{cases} 1 & \text{if won,} \\ -1 & \text{if lost,} \\ 0 & \text{if null or ongoing.} \end{cases}$$

- objective function $\zeta = \sum_t r_r$

Reinforcement Learning

Bellman equation and the TD error [Sutton, 1988]

Bellman approach:

- suppose $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$

Bellman approach:

- suppose $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- We define: the value of a state $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  This quantifies what will happen to the agent in its future if it behaves according to π.

Reinforcement Learning

Bellman equation and the TD error [Sutton, 1988]

Bellman approach:

- suppose $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- We define: the value of a state $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  This quantifies what will happen to the agent in its future if it behaves according to π.
- re-written as: $V(x_t|\pi) = \mathbb{E}(r_t) + \gamma \mathbb{E}(V(x_{t+1}|\pi))$

  sum of what will happen immediately + what will happen then.

Reinforcement Learning

Bellman equation and the TD error [Sutton, 1988]

Bellman approach:

- suppose $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- We define: the value of a state $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  This quantifies what will happen to the agent in its future if it behaves according to π.
- re-written as: $V(x_t|\pi) = \mathbb{E}(r_t) + \gamma\mathbb{E}(V(x_{t+1}|\pi))$

  sum of what will happen immediately + what will happen then.
- $V(x_t|\pi^*) = \max_a(\mathbb{E}(r_t) + \gamma\mathbb{E}(V(x_{t+1}|\pi^*)))$

Reinforcement Learning

Bellman equation and the TD error [Sutton, 1988]

Bellman approach:

- suppose $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- We define: the value of a state $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  This quantifies what will happen to the agent in its future if it behaves according to π.
- re-written as: $V(x_t|\pi) = \mathbb{E}(r_t) + \gamma \mathbb{E}(V(x_{t+1}|\pi))$

  sum of what will happen immediately + what will happen then.
- $V(x_t|\pi^*) = \max_a(\mathbb{E}(r_t) + \gamma \mathbb{E}(V(x_{t+1}|\pi^*)))$
- $r_t + \gamma(V(x_{t+1}|\pi)) - V(x_t|\pi)$

  is an estimation of the error of estimation of V: TD-error

  This TD-error may be used to learn the optimal behavior.

- computing V by gradient descent:

$$V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$$

Reinforcement Learning

The temporal difference

- computing V by gradient descent:

$$V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$$

- We may also define the value of an (x, a) pair (also known as its quality):

$$Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$$

- computing V by gradient descent:

  $$V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$$
- We may also define the value of an (x, a) pair (also known as its quality):

  $$Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$$
- $\rightsquigarrow$
  $$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \eta[r_t + \gamma \max_b Q(x_{t+1}, b) - Q(x_t, a_t)]$$

Reinforcement Learning

The temporal difference

- computing V by gradient descent:

  $$V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$$
- We may also define the value of an (x, a) pair
  (also known as its quality):

  $$Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$$
- $\rightsquigarrow$

  $$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \eta[r_t + \gamma \max_b Q(x_{t+1}, b) - Q(x_t, a_t)]$$
- $\rightsquigarrow$ learning π^* algorithm.

Idea of an RL algorithm:

1. initialize the agent with an *e.g.* random policy
2. set the agent in some random initial state
3. run the agent in the environment
4. at each step, record the state, the action performed, the reward collected, and the next state
5. at some point, use this information to fit an estimate of Q
6. when task fullfilled or takes too much time, go to step 2.

Reinforcement Learning

Monte Carlo approach

Idea of an RL algorithm:

1. initialize the agent with an *e.g.* random policy
2. set the agent in some random initial state
3. run the agent in the environment
4. at each step, record the state, the action performed, the reward collected, and the next state
5. at some point, use this information to fit an estimate of Q
6. when task fullfilled or takes too much time, go to step 2.

At step 5, the TD error is used as the quantity to minimize.

Idea of an RL algorithm:

1. initialize the agent with an *e.g.* random policy
2. set the agent in some random initial state
3. run the agent in the environment
4. at each step, record the state, the action performed, the reward collected, and the next state
5. at some point, use this information to fit an estimate of Q
6. when task fullfilled or takes too much time, go to step 2.

At step 5, the TD error is used as the quantity to minimize.
This is the essence of Q-Learning.

Reinforcement Learning

Q-Learning [Watkins, 1989]

- $Q(x, a) \leftarrow$ some value (0, random, ...)
- $t \leftarrow 0$
- Initialize the state of the agent x_t
- **while** episode not completed, **do:**
  - choose an action to perform in state x_t: a_t
  - perform this action and observe r_t and x_{t+1}
  - update $Q(x, a)$:

$$\begin{aligned} Q(x_t, a_t) &\leftarrow Q(x_t, a_t) + \alpha \text{ TD-error} \\ &\leftarrow Q(x_t, a_t) + \alpha[r_t + \max Q(x_{t+1}, b_b - Q(x_t, a_t))] \end{aligned}$$

  - $t++$

- $Q(x, a) \leftarrow$ some value (0, random, ...)
- **repeat**
  - $t \leftarrow 0$
  - Initialize the state of the agent x_t
  - **while** episode not completed, **do:**

    ...

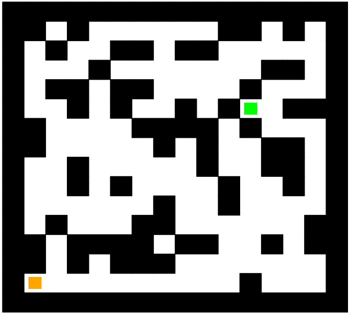
  1 episode
- **until** some stopping criterion is met.

At the completion of this algorithm (if you looped enough):
$\pi^*(x) = \arg\max_a Q(x, a), \forall x$

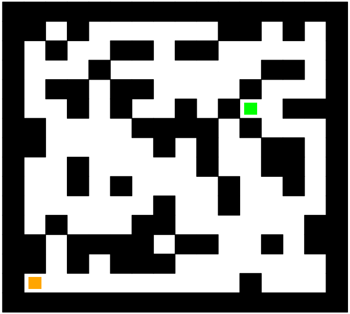
Reinforcement Learning

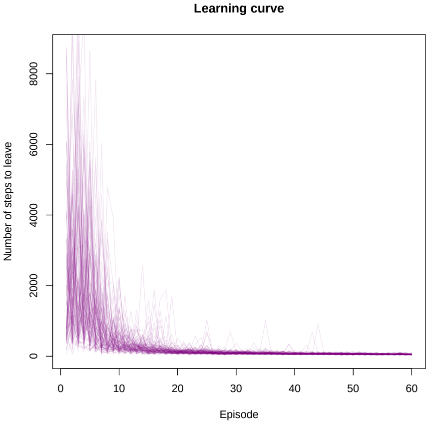
Q-Learning in action

We use an extremely basic Q-Learning.
Has a very local perception: sees only the 4 neighboring cells.

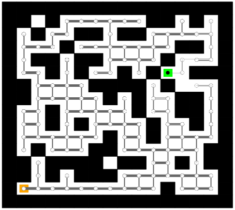


We use an extremely basic Q-Learning.
Has a very local perception: sees only the 4 neighboring cells.





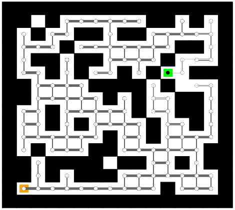
1st reach



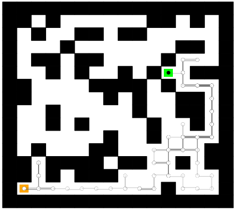
Reinforcement Learning

Q-Learning in action

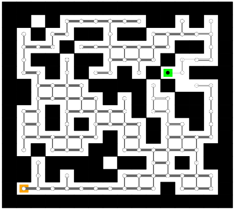
1st reach



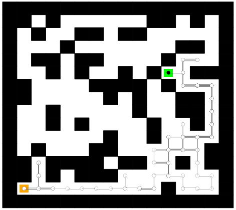
10th reach



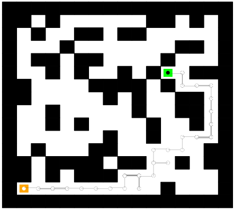
1st reach



10th reach



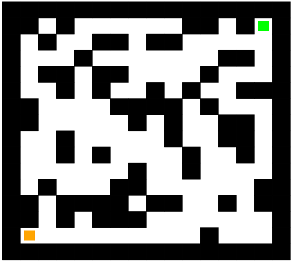
60th reach

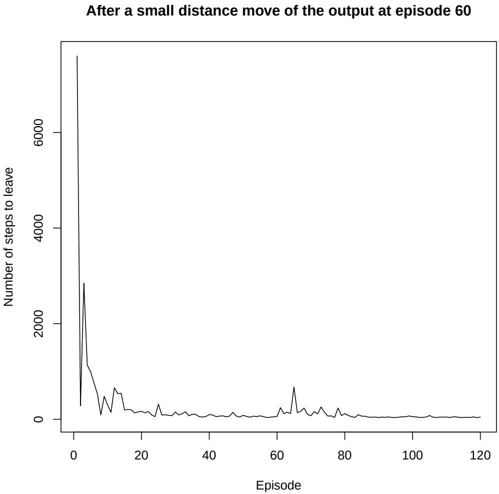


Reinforcement Learning

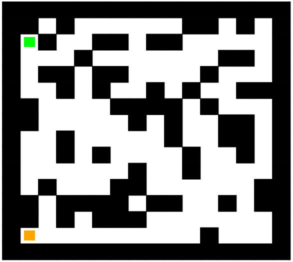
Q-Learning continuously adapts to its environment

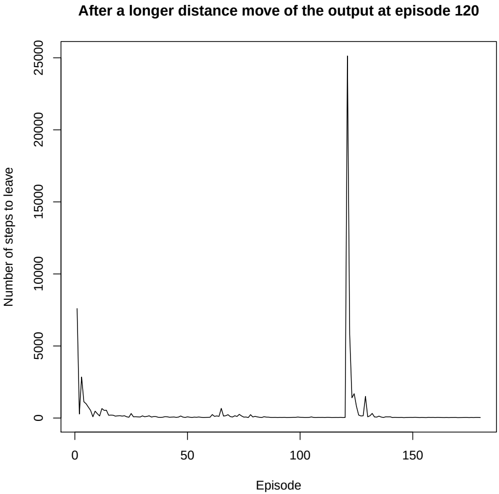
The goal state moves nearby:



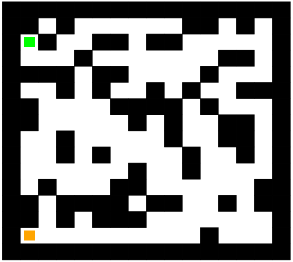


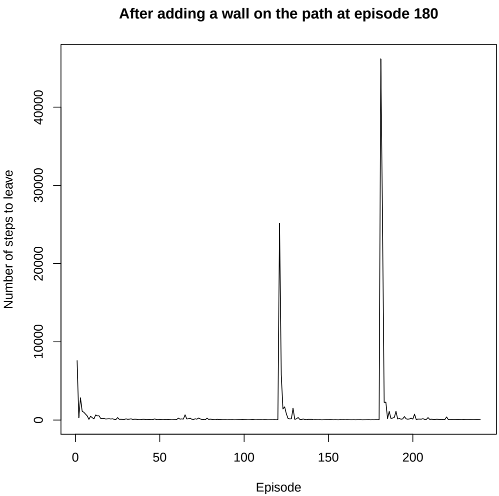
The goal state moves farther away:





Blocking the path:





- This is the "tabular" Q-Learning: Q is represented in a "table".

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?

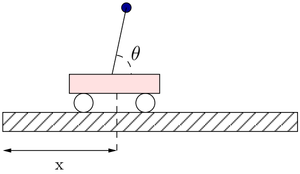
- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.

- This is the “tabular” Q-Learning: Q is represented in a “table”.
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.
- `Q (x, a)` returns an estimate of $Q(x, a)$.

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.
- `Q (x, a)` returns an estimate of $Q(x, a)$.
- This estimate may be updated/improved by learning.

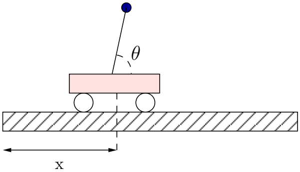
Reinforcement Learning
Value function



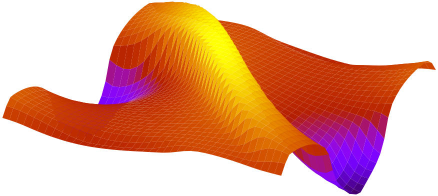
State is $(\theta, \dot{\theta})$
Action is $\ddot{\theta}$

Reinforcement Learning

Value function



State is $(\theta, \dot{\theta})$
Action is $\ddot{\theta}$



$(\theta, \dot{\theta})$ plane
z is $V(x)$
Maximize value $\rightsquigarrow$ reach the top of V

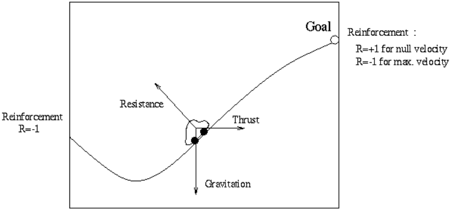
Reinforcement Learning

Handling large $\mathcal{X}$: the function approximator zoo

- neural network [Lin, 1991; Riedmiller, 2005; ...],
- random forest [Geurts *et al.*, 2006],
- SVM and kernels,
- and many other ideas for statistical learning (supervised learning).
- Tabular with progressive and adaptive state partitioning.

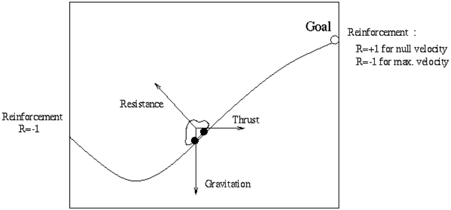
Reinforcement Learning

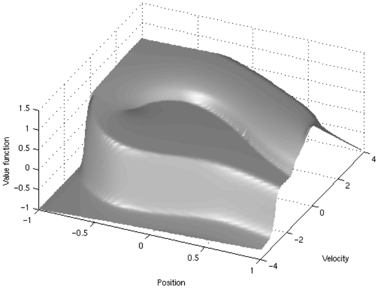
Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]



Reinforcement Learning

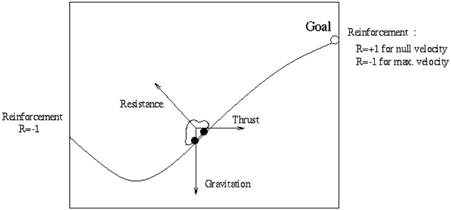
Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

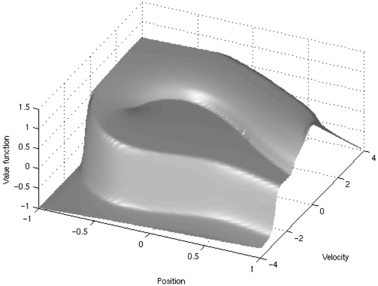


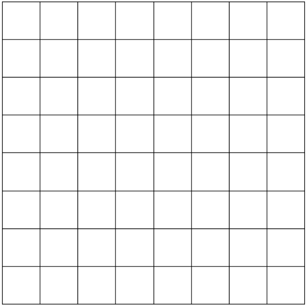


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

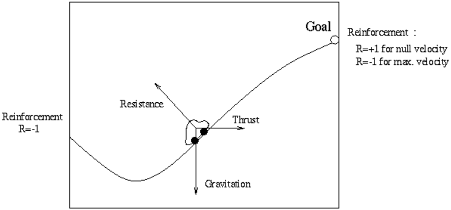


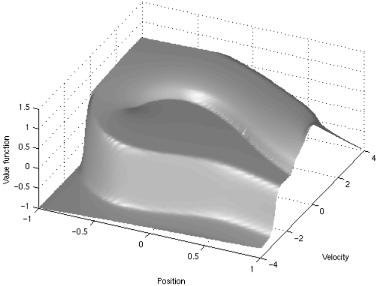


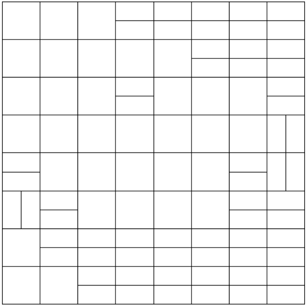


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

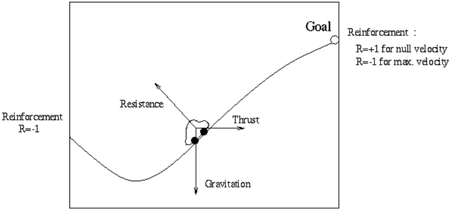


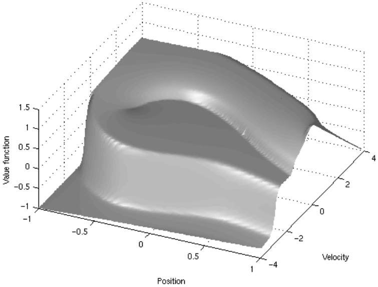


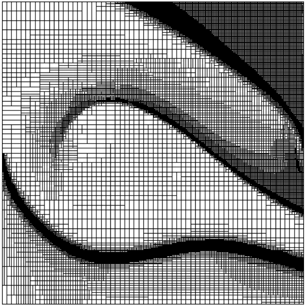


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

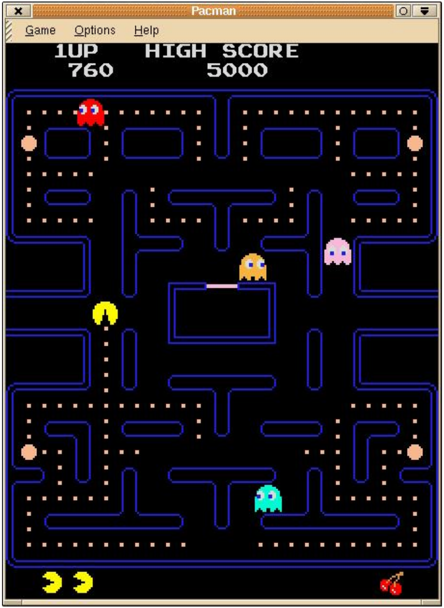


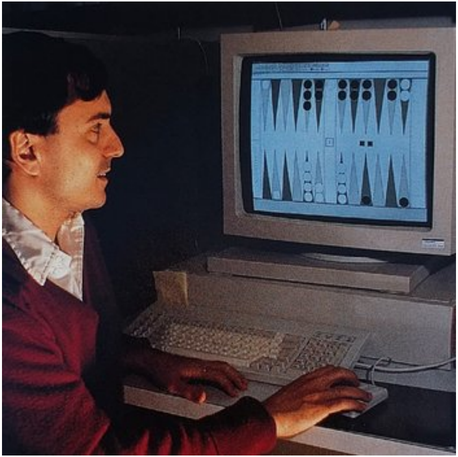




Reinforcement Learning

Applications



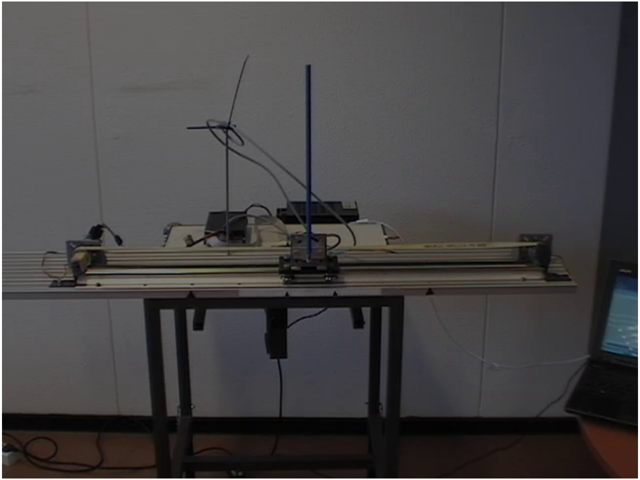


- Backgammon is studied at least since 1974
- Branching factor: 800
- TD-Gammon: "successor of Neurogammon, trained by superviser learning. NeuroGammon won the 1st Computer Olympiad in London in 1989, handily defeating all opponents. Its level of play was that of an intermediate-level human player." (Source: wikipedia)
- raw representation of the board position
- trained with TD(λ) algorithm
- no knowledge, self-play
- hand-crafted features
- 3-plies in v3

Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995

Reinforcement Learning

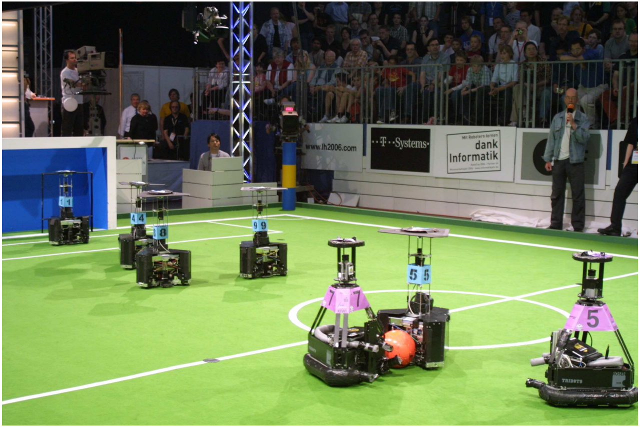
Application to robotics



Riedmiller, Neural reinforcement learning to swing-up and balance a real pole, *Proc. 2005 IEEE International Conference on Systems, Man and Cybernetics*

Reinforcement Learning

Application to robotics



Lauer et al., Cognitive Concepts in Autonomous Soccer Playing Robots, *Cognitive Systems Research*, **11**(3), 287:309, September, 2010
(No Deep Learning! only shallow multi-layer perceptron)

Reinforcement Learning

Application: Guesswhat?!

- Learning to dialog in natural language by RL.
- Usually, spoken dialog systems are rule based, or trained by supervised learning.

Reinforcement Learning

Application: Guesswhat?!

- 2 player game: oracle and guesser
- oracle: assigned an object in a picture
- guesser: has to locate the object by asking yes-no questions to the oracle, which has to answer correctly the questions.
- formulated as an RL problem

Is it a person?	No
Is it an item being worn or held?	Yes
Is it a snowboard?	Yes
Is it the red one?	No
Is it the one being held by the person in blue?	Yes

Is it a cow?	Yes
Is it the big cow in the middle?	No
Is the cow on the left?	No
On the right ?	Yes
First cow near us?	Yes

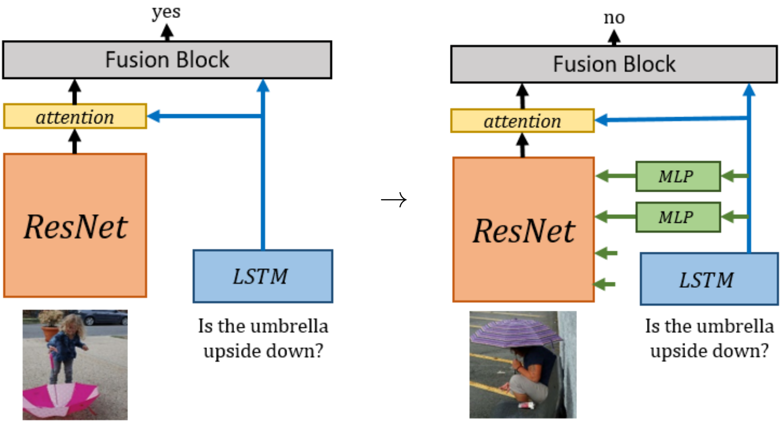
- Trained on 70k images, 134 k unique objects, 800k Q&A pairs

- End-to-end: from raw pngs to dialogs
- RL performs better than supervised learning
- Learning to dialog through a picture
- Deep reinforcement learning
- Combines vision + language: Resnet + LSTM

Reinforcement Learning

Application: Guesswhat?!: Modulating vision by language, ...

- Vision is modulated by language:

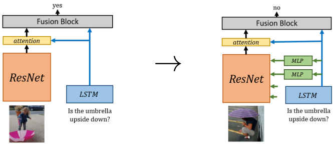


`https://guesswhat.ai/`

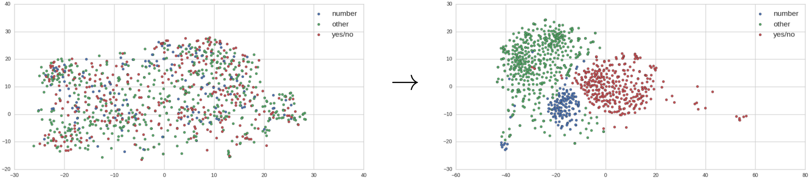
Reinforcement Learning

Application: Guesswhat?!: Modulating vision by language, ...

- Vision is modulated by language:



- Modulation improve embedding:



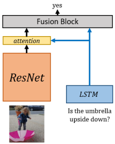
`https://guesswhat.ai/`

Partially funded by IGLU Interactive Grounded Language Understanding in collaboration with MILA, and researchers from Deepmind, and Google Brain.

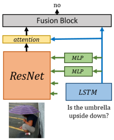
Reinforcement Learning

Application: Guesswhat?!: Modulating vision by language, ...

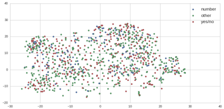
- Vision is modulated by language:



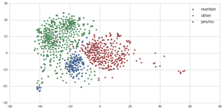
→



- Modulation improve embedding:



→



- Modulation may be used beyond vision: any "signal" might be modulated.

`https://guesswhat.ai/`

Partially funded by IGLU Interactive Grounded Language Understanding in collaboration with MILA, and researchers from Deepmind, and Google Brain.

Reinforcement Learning

Application: board games

- Learning to play board games using only the rules of the game
- Alpha Go learned to play Go by using games played by humans
- Alpha Zero learned to play even better by itself by RL.
- then other board games (chess, draughts, reversi, ...)
- then Starcraft II

Reinforcement Learning

Alpha Zero type of algorithms

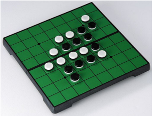
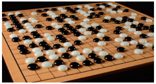
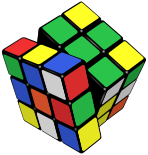
- RL
- $+$ various tricks to stabilize learning and make it more efficient
- MCTS as a key component
- moderately deep network as function approximator

Outro

- learning options
- learning representation
- generalization in RL
- time varying environments
- transfer learning
- life-long learning
- explaination/accountability of the learned behavior

Take home message

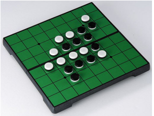
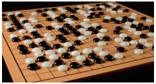
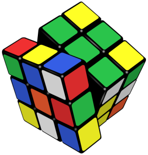
- Many problems can benefit from a sequential decision making point of view.

Not only games.

Take home message

- Many problems can benefit from a sequential decision making point of view.

  Not only games.
- Reinforcement learning outperforms supervised learning.

Bibliography

- Goodfellow *et al.*, *The Deep Learning Book*, MIT Press, 2016, `thedeeplearningbook.org`
- François-Lavet *et al.*, *An Introduction to Deep Reinforcement Learning*, Foundations andTrends in Machine Learning: Vol. 11, No. 3-4., `https://arxiv.org/pdf/1811.12560.pdf`
- Lattimore and Szepesvári, *Bandit algorithms*, CUP 2018, `http://banditalgs.com/`
- Lapan, *Practical Deep Reinforcement Learning*, Packt, 2018
- Sutton and Barto, *Reinforcemnt Learning*, 2nd ed, MIT Press, 2018, `http://incompleteideas.net/book/the-book-2nd.html`

Bibliography

- Maillard, hdr dissertation, 2019
- Mary et al., Bandits and Recommender Systems, Proc. MOD, LNCS 9432, 2015, hal-01256033.
- Preux et al., Bandits attack function optimization, CEC 2014
- Silver *et al.*, Mastering the game of Go without human knowledge, Nature, **550**, 2017
- Silver *et al.*, Mastering the game of Go with deep neural networks and tree search, Nature, **529**, 2016
- Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995
- Valko, hdr dissertation, 2017
- White, Bandit algorithms for website optimization, O'Reilly,

For more, join the

Reinforcement Learning Summer Scool

Villeneuve d'Ascq, 1-12 July

Lectures + practical sessions + keynotes

`rlss.inria.fr`