Reinforcement learning

Marco Piastra

Multi-Armed Bandit

[image from wikipedia]

- Basic definitions

  N arms (i.e. a row of N old-style slot machines)

  Each arm i yields a (stochastic) reward r with probability distribution $P_i(r)$

  Each time t in a sequence, the player (i.e. the agent) selects the arm $i(t)$

    In other words, $i(t)$ is the *strategy* adopted by the agent

- Problem

  Find a strategy $i(t)$ that maximizes the *total reward* over time

    The strategy will include random choices i.e. it will be *stochastic*

  For simplicity, only Bernoulli rewards (i.e. either 0 *or* 1*) will be considered here*

Multi-Armed Bandit: strategies

- Informed strategy

  At all times, select the arm with higher probability of reward:

  $$i(t) = \text{argmax}_i P_i(1)$$

  Clearly, this strategy is optimal but requires knowing all distributions $P_i(r)$

  With enough data (*e.g. from other players*), these distributions can be learnt

- Random strategy

  At all times, select an arm i at random, with uniform probability

  How does the Random strategy compare with the optimal, informed strategy?

Multi-Armed Bandit: evaluating strategies

- *Total Expected Regret*

  How far from optimality a strategy is, considering the total reward over T trials

  A first definition (i.e. total regret with expected rewards)

$$R(T) := T\mu^* - \sum_{t=1}^{T} \mu_{i(t)}$$

where

expected (i.e. *mean*) reward of arm *k*

$$\mu_k := \sum_r P_k(r) \qquad \mu^* := \max_k \mu_k$$

A better definition (*Total Expected Regret*)

$$\overline{R}(T) := T\mu^* - \sum_{k=1}^{N} \mathrm{E}[T_k(T)]\mu_k = \sum_{k=1}^{N} \mathrm{E}[T_k(T)]\Delta_k$$

where

$$\Delta_k := \mu^* - \mu_k$$

number of times arm *k* is selected in T trials (*a random variable*)

With the optimal, informed strategy, the total expected regret is 0.
Whereas, with the *random strategy* the total expected regret grows linearly over time:

$$\overline{R}(T) = \frac{T}{N} \sum_{k=1}^{N} \Delta_k$$

Multi-Armed Bandit: *Online learning*

- Adaptive strategy: *exploration vs. exploitation*

  Typically, in real-world cases, the agent has no previous knowledge about the N arms

  exploration: make trials over the set of *N* arms to learn about the expected reward μ_k

  exploitation: make use of the current best guess about the expected rewards μ_k

- Greedy strategy

  Initialize all the estimated values μ_k at random

  Repeat: select the arm with the current best estimated reward $i = \text{argmax}_k \hat{\mu}_k$

  and update the current estimate about *i* as the *average* reward

$$\hat{\mu}_i := \frac{\sum_{t=1}^{T_i} r_{i,t}}{T_i}$$

reward of arm i at trial t

number of times the arm i has been played

- ε-greedy strategy $(0 < \varepsilon < 1)$

  Initialize all the estimated values μ_k at random

  Repeat: select with probability $(1 - \varepsilon)$ the arm with the current best estimated reward
  or else (*i.e. with probability* ε) select one arm at random
  and update the current estimate about *i* as the *average* reward

Multi-Armed Bandit: *Online learning*

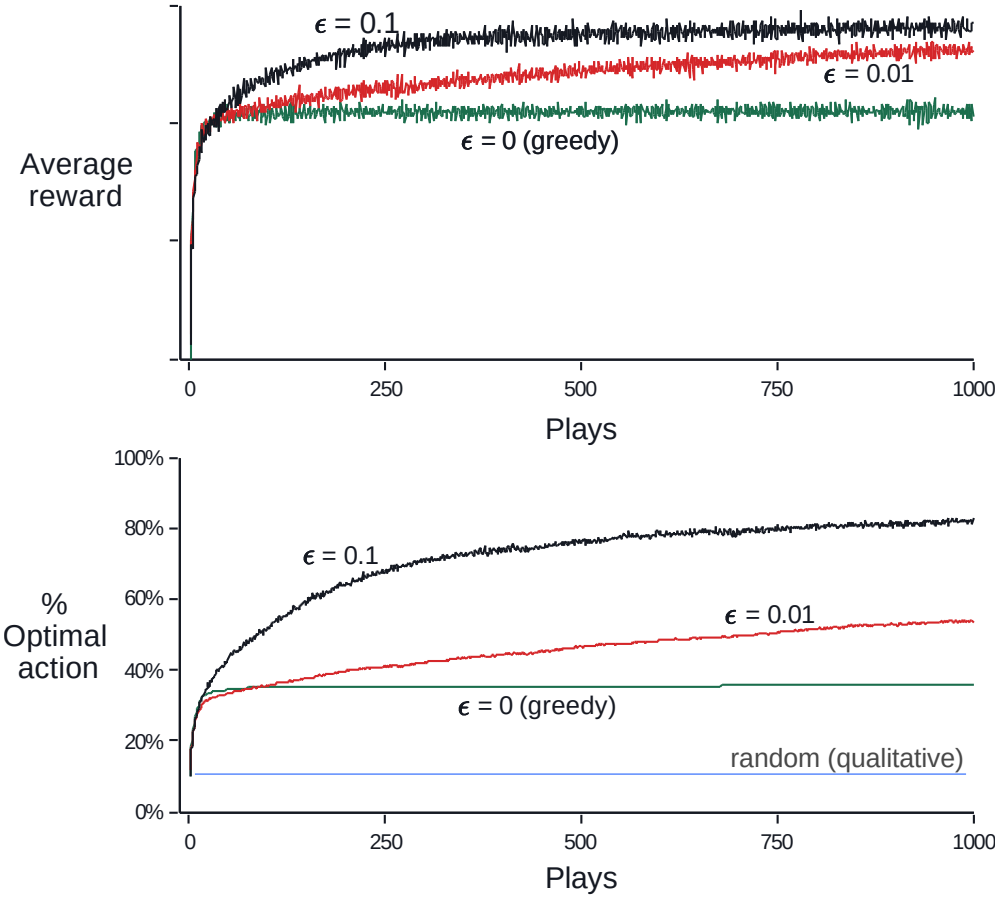
- Experimental comparison of different strategies

  10 arms bandit with different rewards (10-arms testbed)
  Averaged over 2000 runs (i.e. sequences of trials)

  After a certain period of time, the *greedy* strategy stops exploring and exploits its estimates

  whereas, the ε-greedy strategy keeps exploring and approaches optimality

  The random strategy never improves its performances, as expected



Multi-Armed Bandit: evaluating strategies

- *From a theoretical standpoint*

  All *greedy* strategies are *biased*: they depend on the initial random distribution

  Optimistic variant: initially, set all $\hat{\mu}_k := 1$

  The average total regret always grows linearly, in the long run

  In fact:

  - on the average, the *greedy* strategy will get stuck in a suboptimal choice
  - the ε-greedy strategy will continue to choose an arm at random (with probability ε)

  Can we do any better?

Multi-Armed Bandit: Optimal *online learning*

- Lower bound theorem [Lai & Robbins 1985]

  Consider a generic, adaptive (i.e. learning) strategy for the multi-armed bandit problem with Bernoulli reward

$$\lim_{T\to\infty} \overline{R}(T) \geq \ln T \sum_{k|\Delta_k>0} \frac{\Delta_k}{\mathrm{kl}(\mu_k, \mu^*)}$$

  where

$$\mathrm{kl}(\mu_k, \mu^*) := \mu_k \ln \frac{\mu_k}{\mu^*} + (1-\mu_k) \ln \frac{(1-\mu_k)}{(1-\mu^*)}$$

  a special case of the *Kullback-Leibler divergence* :
  it measures of the difference between two (Bernoulli) distributions

In other words, we can achieve logarithmic growth for the total expected regret, but not better: any adaptive strategy must play suboptimal arms a minimum number of times

$$\lim_{T\to\infty} \mathrm{E}[T_k(T)] \geq \frac{\ln T}{\mathrm{kl}(\mu_k, \mu^*)}$$

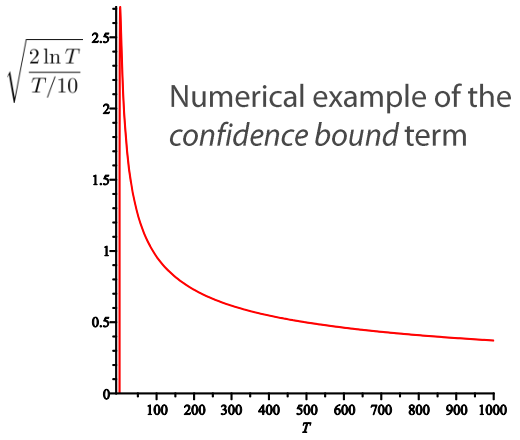
Multi-Armed Bandit: UCB strategy

- Upper confidence bound (UCB) strategy [Auer, Cesa-Bianchi and Fisher 2002]

  Initialize all the estimates of the expected reward $\hat{\mu}_k := 0$
  Play each arm once (*to avoid zeroes in the formula below*)

  Repeat: select the arm $i = \text{argmax}_k \hat{\mu}_k + \sqrt{\frac{2 \ln T}{T_k}}$ — T: total number of trials; T_k: number of times the arm k has been played

  and update the current estimate about i
  as the *average* reward



Numerical example of the *confidence bound* term

Theorem

With the UCB strategy

$$\lim_{T \to \infty} \mathrm{E}[T_k(T)] \leq \frac{8 \ln T}{\Delta_k^2} + c$$

(c: a (small) constant)

where it can be shown that

$$\frac{8}{\Delta_k^2} \geq \frac{1}{\mathrm{kl}(\mu_k, \mu^*)}$$

(*there is a reasonably small gap between the two bounds – i.e. near optimality*)

Multi-Armed Bandit: Thompson Sampling

- Thompson Sampling strategy (*also 'Bayesian Bandit'*) [Thompson, 1933]

  Initialize all the expected reward $\mu_k :\sim \mathrm{Beta}(1,1)$

  i.e. assume that this is a random variable with this (*prior*) distribution

  Repeat:
  sample each of the N distributions to obtain an estimate $\hat{\mu}_k$

  select the arm $i = \mathrm{argmax}_k \hat{\mu}_k$
  and update the *posterior* distribution

  $$\mu_i :\sim \mathrm{Beta}(R_i + 1,\ T_i - R_i + 1)$$

  T_i: number of times the arm has been played

  R_i: total (*Bernoulli*) reward from this arm (*i.e. number of wins*)

  Theorem [Kaufmann et al., 2012]

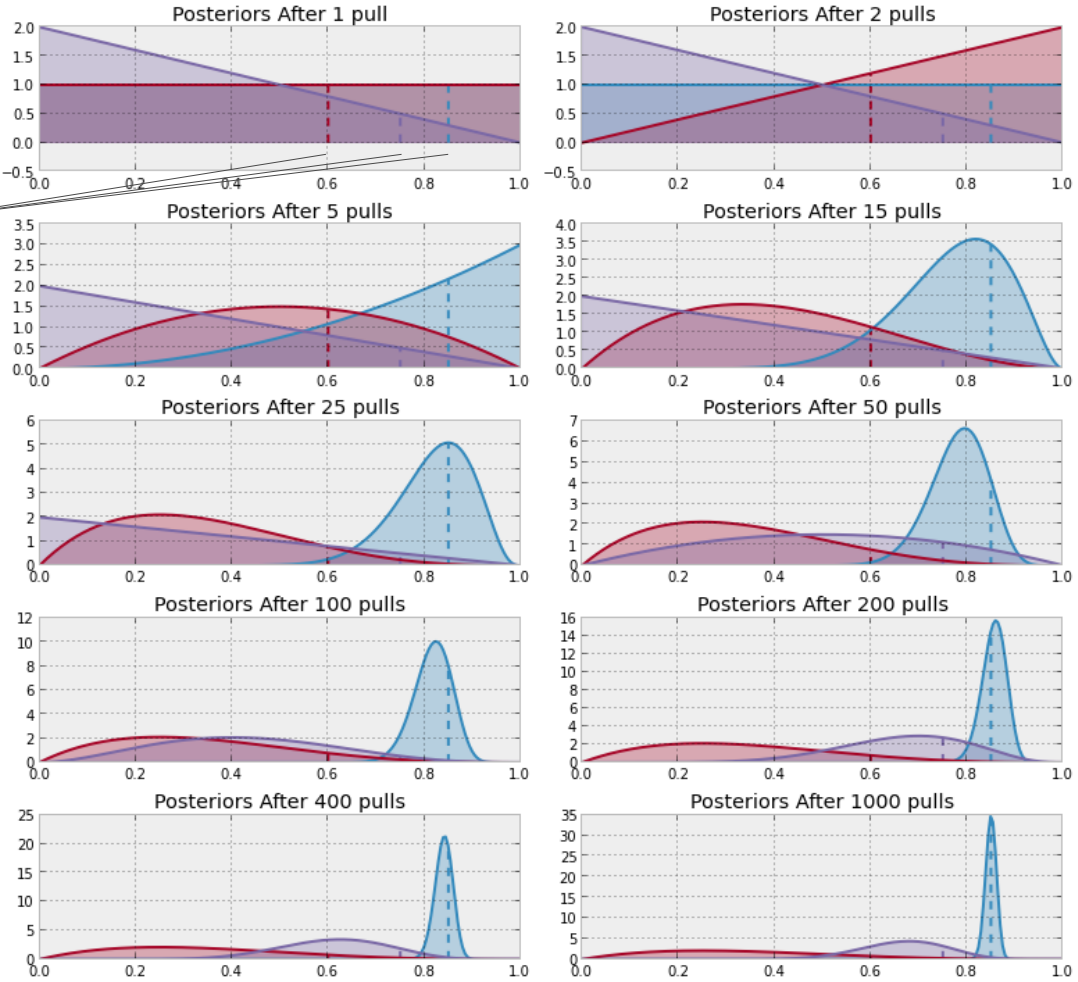
  The Thompson Sampling strategy has essentially the same theoretical bounds of the UCB strategy

Multi-Armed Bandit: Thompson Sampling

- Thompson Sampling strategy (*also 'Bayesian Bandit'*) [Thompson, 1933]

  Example run with 3 arms: trace of the posterior probabilities for each μ_k

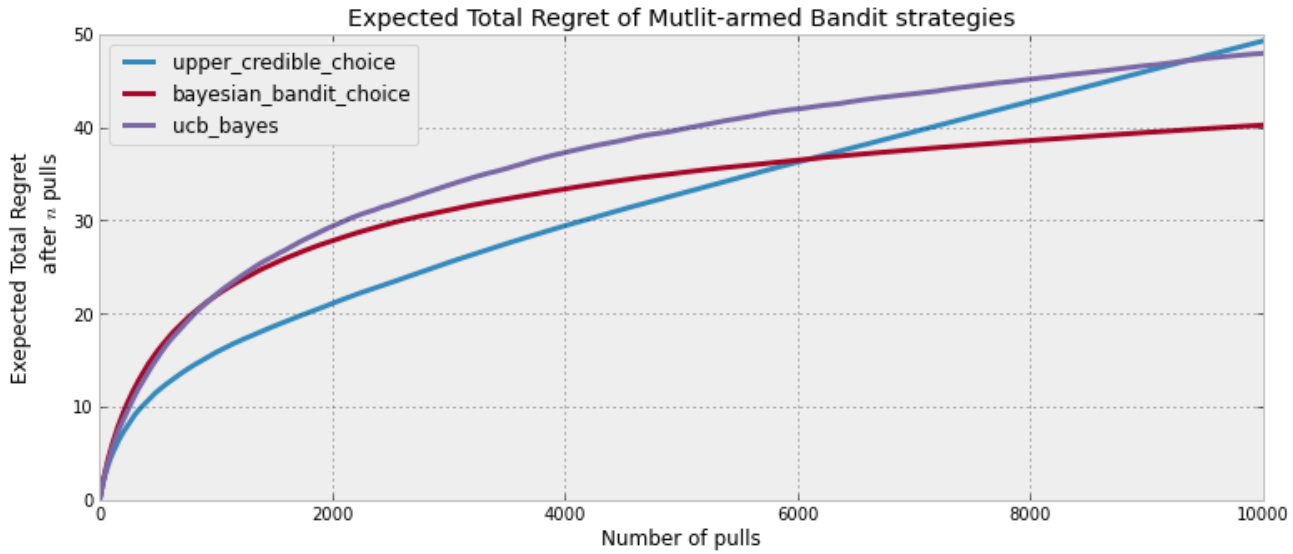
'true' means



Multi-Armed Bandit: Thompson Sampling

- Thompson Sampling strategy (*also 'Bayesian Bandit'*) [Thompson, 1933]

  In practical experiments, this strategy shows better performances in the long run
  [Chapelle & Li, 2011]

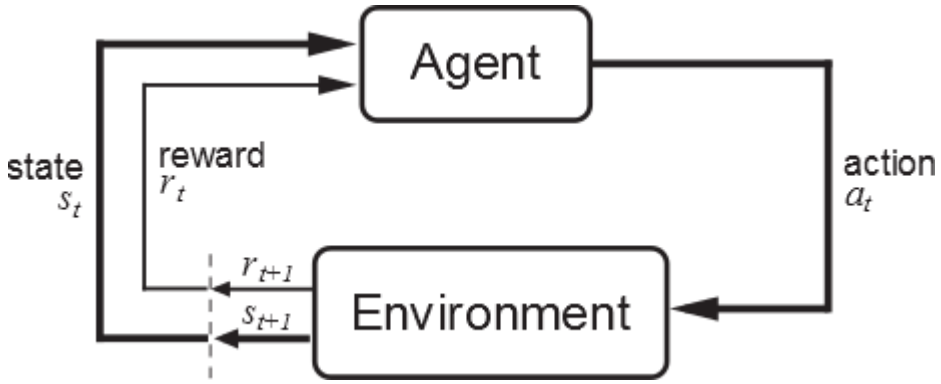


Actually, Thompson Sampling is a preferred strategy at Google Inc.
(see https://support.google.com/analytics/answer/2846882?hl=en)

Agent/Environment Interactions

With multi-armed bandits, the context never changes
*in the sense that the optimal choice does **not** depend on the current state*

What if the actions of the agent change the state of its interaction with the environment?



Examples:

- a_t could be a *move in a game*, whereby the agent changes the state of the game
- a_t could be a *movement*, whereby the agent changes its position in the environment

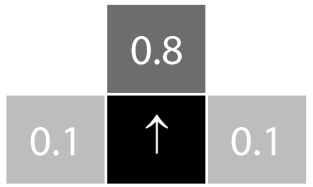
An example: *gridworld*

	1	2	3	4
1	-0.02	-0.02	-0.02	1
2	-0.02		-0.02	-1
3	-0.02	-0.02	-0.02	-0.02

The *state* of the agent is the position on the grid: e.g. (1,1), (3,4), (2,3)

At each time step, the agent can *move* one box in the directions ←↑↓→

The effect of each move is somewhat stochastic, however: for instance a move ↑ has a slight probability of producing a different (*and perhaps unwanted*) effect

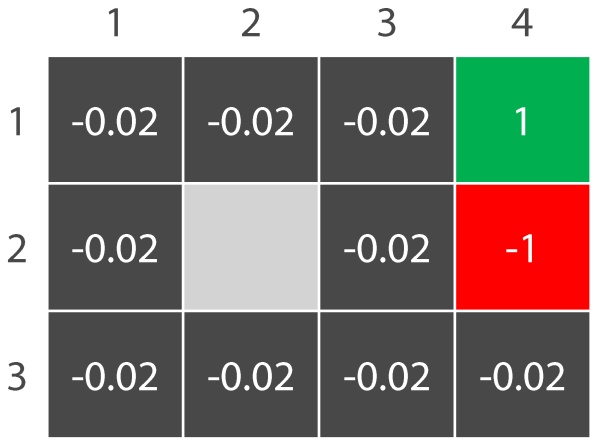


Entering each state yields the *reward* shown in each box above

There are two *absorbing states*: entering either the green or the red box means exiting the *gridworld* and completing the game

- What is the best (*i.e. maximally rewarding*) movement policy?

Markov Decision Process (MDP)



Formalization and abstraction of the gridworld example

Markov Decision Process: $< S, A, r, P, \gamma >$

A set of *states* : $S = \{s_1, s_2, \dots\}$

A set of *actions* : $A = \{a_1, a_2, \dots\}$

A *reward function* : $r : S \to \mathbb{R}$

A *transition probability distribution* : $P(S_{t+1} \mid S_t, A_t)$

Markov property: the transition probability depends only the previous state and action

A *discount factor* (see after): $0 \leq \gamma \leq 1$

Markov Decision Process (MDP): policies and values

The agent is supposed to adopt a deterministic *policy* : $\pi : S \rightarrow A$

In other word, the agent always chooses its *action* depending on the *state* alone

The *value function* for each policy π is defined as:

$$V^{\pi}(S_t) := \mathrm{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \cdots \mid \pi]$$

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards are weighted less (by the agent) than immediate ones

Note also that all states beyond S_t must be described by *random variables* : the policy is deterministic but the state transition is not

In the *gridworld* example:

- The set of states is finite
- The set of actions is finite
- For every policy, each entire story is finite

  Sooner or later the agent will fall into one of the *absorbing states*

Bellman equations

By working on the definition of value function:

$$\begin{aligned} V^{\pi}(S_t) &:= \mathrm{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \cdots \mid \pi] \\ &= \mathrm{E}[r(S_t) + \gamma (r(S_{t+1}) + \gamma r(S_{t+2}) + \cdots) \mid \pi] \\ &= \mathrm{E}[r(S_t) + \gamma V^{\pi}(S_{t+1}) \mid \pi] \end{aligned}$$

This can be rewritten as:

$$V^{\pi}(S_t) = r(S_t) + \gamma \sum_{S_{t+1}} P(S_{t+1} | S_t, \pi(S_t)) V^{\pi}(S_{t+1})$$

This is true for any *state*, so there is one such equation for each of those

There are exactly $|S|$ (linear) Bellman equations for $|S|$ variables:
in general, given π, V^{π} can be computed in closed form

Optimal policy – Optimal value function

- Basic definitions

$$\pi^*(s) := \text{argmax}_\pi V^\pi(s),\ \forall s \in S$$

$$V^*(s) := \max_\pi V^\pi(s),\ \forall s \in S$$

Property: for every MDP, there exists such an optimal deterministic policy (*possibly non-unique*)

With Bellman Equations:

$$\max_\pi V^\pi(S_t) = r(S_t) + \gamma \max_\pi (\sum_{S_{t+1}} P(S_{t+1}|S_t, \pi(S_t)) V^\pi(S_{t+1}))$$

$$V^*(S_t) = r(S_t) + \gamma \max_\pi (\sum_{S_{t+1}} P(S_{t+1}|S_t, \pi(S_t)) V^*(S_{t+1}))$$

$$= r(S_t) + \gamma \max_a (\sum_{S_{t+1}} P(S_{t+1}|S_t, a) V^*(S_{t+1}))$$

Therefore:

$$\pi^*(S_t) := \text{argmax}_a (\sum_{S_{t+1}} P(S_{t+1}|S_t, a) V^*(S_{t+1}))$$

Computing V^* directly from these equations is unfeasible, however

There are in fact $|S|^{|A|}$ possible strategies

However, once V^* has been determined, π^* can be determined as well

Optimal value function: value iteration

- Value iteration algorithm

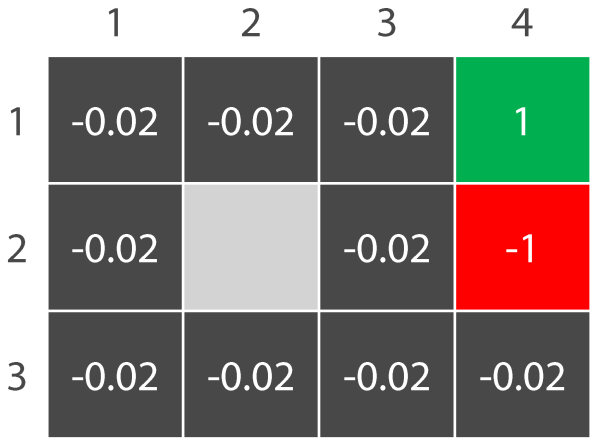
  Initialize: $V(s) := 0, \ \forall s \in S$

  Repeat:

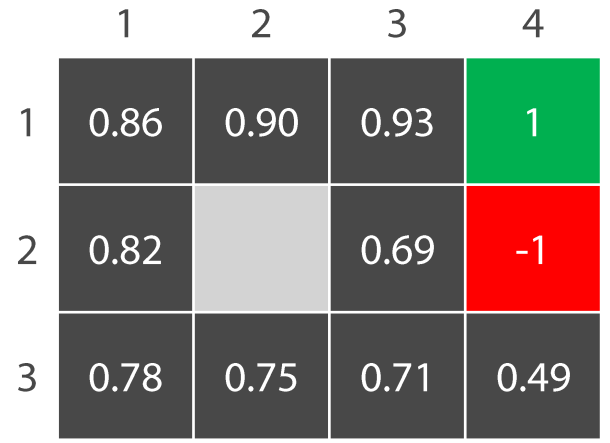
  For every state, update: $V(s) := r(s) + \max_a \gamma \sum_{s'} P(s' \mid s, a) V(s')$

  Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to V^*

Value iteration and optimal policy

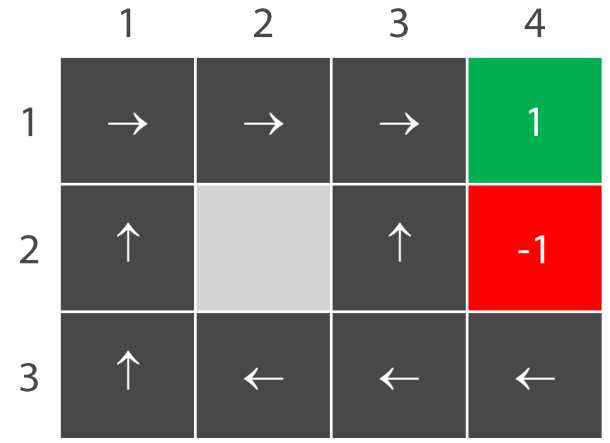


Initial state (plus rewards)



V^*

In each state, compute $\pi^*(S_t) := \text{argmax}_a(\sum_{S_{t+1}} P(S_{t+1}|S_t, a)V^*(S_{t+1}))$



π^*

Optimal policy: policy iteration

- Policy iteration algorithm

  Initialize π at random

  Repeat:

  For each state, compute: $V(s) := V^{\pi}(s)$

  For each state, define: $\pi(s) := \text{argmax}_a \sum_{s'} P(s' \mid s, a) V(s')$

  Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to π^*

  As with the value iteration algorithm, this algorithm uses partial estimates to compute new estimates

  In each iteration after initialization, a greedy policy is selected: the one that maximizes the reward based on current estimates

Optimal policy: policy iteration

- Policy iteration algorithm

  Initialize π at random

  Repeat:

  For each state, compute: $V(s) := V^{\pi}(s)$

  For each state, define: $\pi(s) := \text{argmax}_a \sum_{s'} P(s' \mid s, a) V(s')$

  Theorem: for every fair way (*i.e. giving an equal chance*) of visiting the states in S, this algorithm converges to π^*

  As with the value iteration algorithm, this algorithm uses partial estimates to compute new estimates

  In each iteration after initialization, a greedy policy is selected: the one that maximizes the reward based on current estimates

Learning a model online

So far, we assumed knowledge of the model of the process, that is $P(S_{t+1} \mid S_t, A_t)$
How can we learn it from experience?

Intuitive idea: given *fair* experimental data, the maximum likelihood estimation is

$$P(s' \mid s, a) = \frac{N_{s,a \to s'}}{N_{s,a}}$$

$N_{s,a \to s'}$: number of times action a in state s led to state s'

$N_{s,a}$: number of times action a has been taken in state s

- Policy iteration algorithm

  Initialize π at random

  Repeat:

  Execute policy π for some number of trials

  Update the estimates $P(S_{t+1} \mid S_t, A_t)$

  Apply the value iteration algorithm and get the estimate $\hat{V}^*(s)$

  Update the policy to be greedy w.r.t. $\hat{V}^*(s)$

$$\pi(s) := \text{argmax}_a \sum_{s'} P(s' \mid s, a) \hat{V}^*(s')$$