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 <u>*total reward*</u> 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 (i.e. *optimal*) strategy

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

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

  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

For <u>one</u> sequence of T trials, the *total regret with expected rewards* is

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

decision taken at step t

where

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

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

In a more general definition, the *Total Expected Regret* is

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

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:

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

…since, with a random strategy, $\mathrm{E}[T_k(T)] = \frac{T}{N}$

Multi-Armed Bandit: *Online learning*

Adaptive strategy: *exploration vs. exploitation*

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:

1) select the arm with the current best estimated reward $i = \mathrm{argmax}_k \hat{\mu}_k$
2) 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

Total 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:

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

Multi-Armed Bandit: *Online learning*

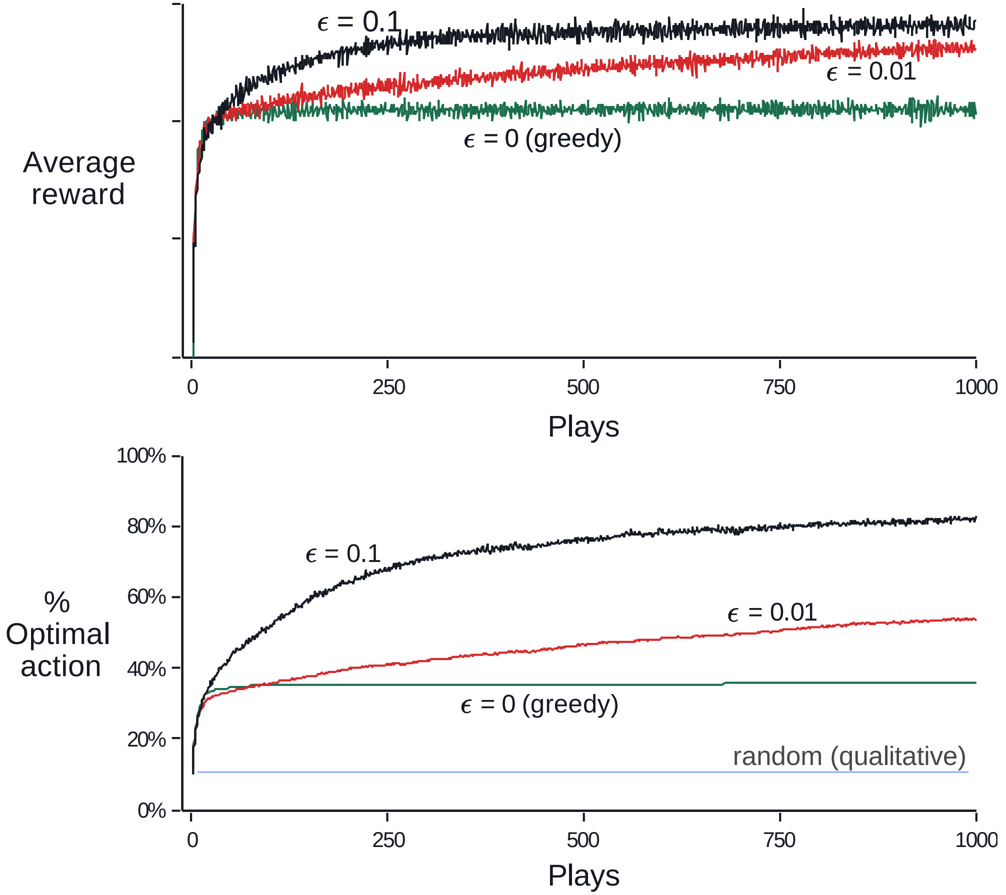
- Experimental comparison of different strategies

  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



1998

Rein

Bart

[imag

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* :
in this case, 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)*

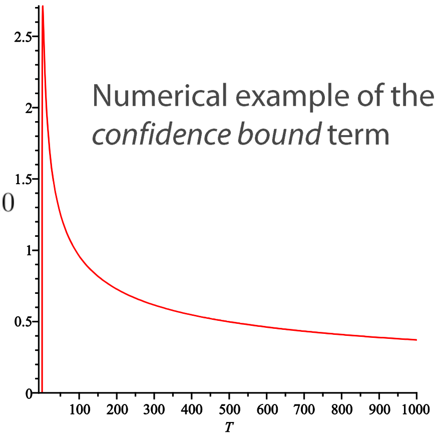
Repeat:

1) select the arm $i = \text{argmax}_k \left(\hat{\mu}_k + \sqrt{\frac{2 \ln T}{T_k}} \right)$

total number of trials

number of times
the arm *k* has been played

2) update the current estimate about *i*
as the *average* reward



$\sqrt{\frac{2 \ln T}{(T/N)}}, \quad N = 10$

Numerical example of the *confidence bound* term

Theorem

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

i.e. a (small) constant

where it can be shown that $\frac{8}{\Delta_k^2} \geq \frac{1}{\mathrm{kl}(\mu_k, \mu^*)}$

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

Multi-Armed Bandit: Thompson Sampling

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

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

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

Repeat:

1) sample each of the N distributions to obtain an estimate $\hat{\mu}_k$
2) select the arm $i = \mathrm{argmax}_k \hat{\mu}_k$
3) update the *posterior* distribution

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

total number of times the arm has been played

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

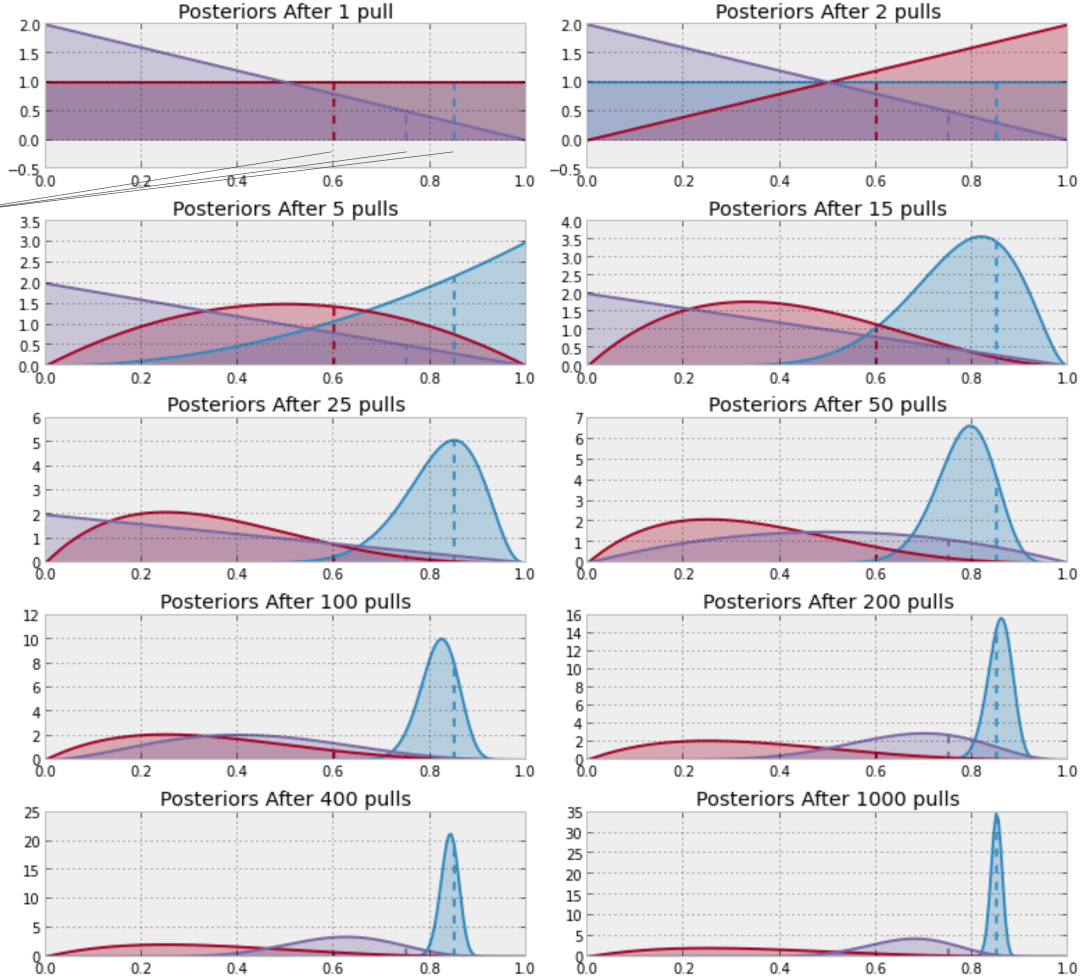
Theorem [Kaufmann et al., 2012]

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

Posteriors After 1 pull

Posteriors After 2 pulls

Posteriors After 5 pulls

Posteriors After 15 pulls

Posteriors After 25 pulls

Posteriors After 50 pulls

Posteriors After 100 pulls

Posteriors After 200 pulls

Posteriors After 400 pulls

Posteriors After 1000 pulls

ban

arm

://ca

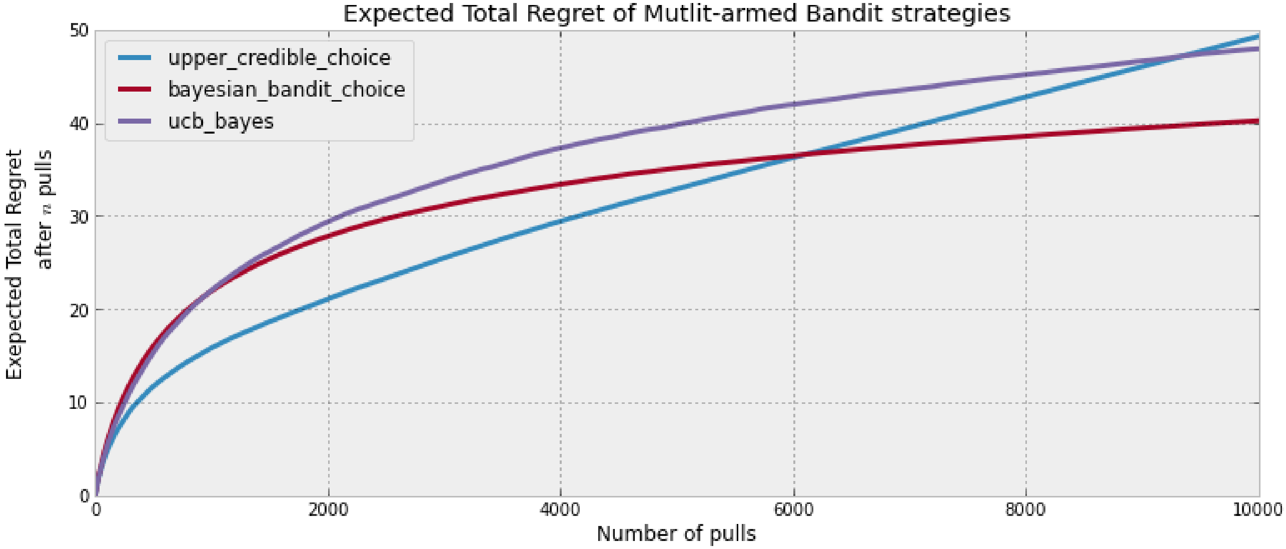
: http

imag

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)

banc

arme

://ca

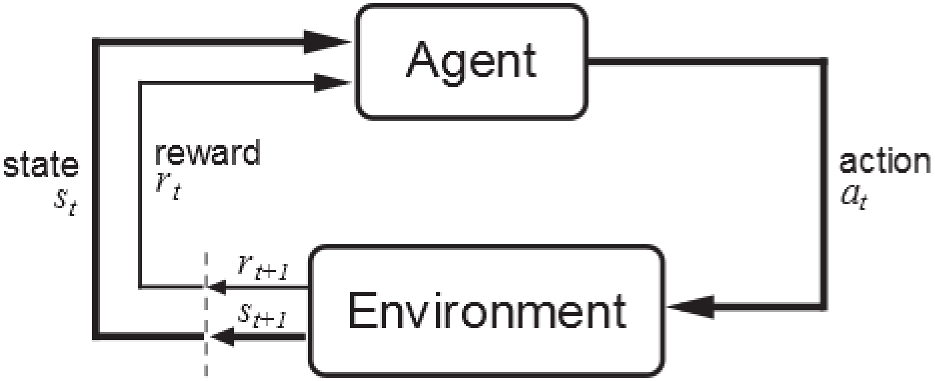
: http

imag

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?



1998

Rein
,

Bart

[imag

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

The agent could be wanting to learn an *optimal strategy* towards a given goal…

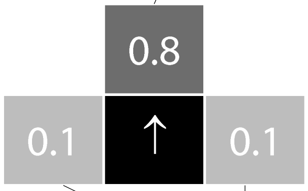
An example: *gridworld*

	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 example, a move ↑ has a slight probability of producing a different (*and perhaps unwanted*) effect



with probability 0.8 the agent will end up here

but with probability 0.2 it might end up here

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)

	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

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* : $0 \leq \gamma \leq 1$

Markov Decision Process (MDP): policies and values

The agent is supposed to adopt a deterministic <u>*policy*</u> : $\pi : S \to A$

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

Given a policy π , the *value function* for is defined, for each state s as:

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

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 S_t must be described by *random variables* :
i.e. 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 <u>finite</u>

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

Bellman equations

By working on the definition of value function:

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

This step requires the 'Law of total expectation' (see wikipedia)

Given that this is a Markov Decision Process, we obtain:

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

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:

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

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

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

Therefore:

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

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:

Note that there is no policy: all actions must be explored

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

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

Value iteration and optimal policy

	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

Initialize states
(e.g. using rewards as initial values)

Iterate and compute

V^*

$\rightarrow$

	1	2	3	4
1	0.86	0.90	0.93	1
2	0.82		0.69	-1
3	0.78	0.75	0.71	0.49

V^*

$\downarrow$

Define the optimal policy as:

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

	1	2	3	4
1	$\rightarrow$	$\rightarrow$	$\rightarrow$	1
2	$\uparrow$		$\uparrow$	-1
3	$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

π^*

Optimal policy: policy iteration

- Policy iteration algorithm

Initialize π at random
Repeat:

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

This step is computationally expensive:
either solve the equations or use value iteration
(with fixed policy)

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.
It is also <u>greedy</u>, in the sense that it exploits its current estimate $V^{\pi}(s)$

Policy iteration converges with very few number of iterations,
but every iteration takes much longer time than that of *value iteration*

The tradeoff with value iteration is the <u>action space</u>:
when action space is large and state space is small, policy iteration could be better

Offline vs. *Online* learning

- *Value iteration* and *policy iteration* are offline algorithms

  The *model*, i.e. the Markov Decision Process is known

  What needs to be learn is the optimal policy π^*

  In the algorithms, *visiting states* just means considering: there is no agent actually playing the game.

- Different conditions: learn by doing

  Suppose the model (i.e. the MDP) is NOT known, or perhaps known only in part

  Then the agent must learn by doing…

Q-Learning

An analogous of the value function V^π

Given a policy π, the *action-value function* is defined, for each pair $< s, a >$ as:

$Q^\pi(s,a) := \sum_{S_{t+1}} P(S_{t+1}|s,a) \cdot V^\pi(S_{t+1})$ *i.e. choose* a *in* s *and then follow* π *afterwards*

Following a similar line of reasoning as before, the optimal action-value function is

$$Q^*(s,a) = \sum_{S_{t+1}} P(S_{t+1}|s,a) \cdot [r(S_{t+1}) + \gamma \max_{a'} Q^*(S_{t+1}, a')]$$

- Q-learning algorithm

Initialize $\hat{Q}(s,a)$ at random, put the agent is in a random state s
Repeat:

1) Select the action $\operatorname{argmax}_a \hat{Q}(s,a)$ with probability $(1-\varepsilon)$
otherwise, select a at random

2) The agent is now in state s' and has received the reward r

3) Update $\hat{Q}(s,a)$ by

$$\Delta\hat{Q}(s,a) = \alpha(r + \gamma \max_{a'} \hat{Q}(s',a') - \hat{Q}(s,a))$$

Q-Learning

- Q-learning algorithm

Theorem (Watkins, 1989): in the limit of that each action is played infinitely often and each state is visited infinitely often and $\alpha \to 0$ as experience progresses, then

$$\hat{Q}(s,a) \to Q^*(s,a)$$

with probability 1

The Q-learning algorithm bypasses the MDP entirely,
in the sense that the optimal strategy is learnt without learning the model