

Artificial Intelligence

Reinforcement learning

Marco Piastra

Multi-Armed Bandit



A row of N old-style slot machines

[image from wikipedia]

■ Basic definitions

N arms or *bandits*

Each arm a yields a random reward r with probability distribution $P(r | a)$

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

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

In other words, π is the *policy* adopted by the agent

■ Problem

Find a policy π that maximizes the *total reward* over time

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

Multi-Armed Bandit: strategies

- Informed (i.e. *optimal*) strategy

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

$$\pi^*(t) = \operatorname{argmax}_a P_i(r = 1 | a)$$

Clearly, this strategy is optimal but requires knowing all distributions $P(r | a)$

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

- Random strategy

At all times, select a bandit a at random, with *uniform probability*

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

Multi-Armed Bandit: basic definitions

- *Actions, Rewards*

$a \in \mathcal{A}$ in this case $a \in \{1, \dots, N\}$

$r \in \mathcal{R}$ in this case $r \in \{0, 1\}$

- *Probability distribution (unknown)*

$P(R | A)$ the probability of reward R for action A (i.e. two random variables)

- *Policy*

$\pi : \mathbb{N}^+ \rightarrow \mathcal{A}$ at each time, defines which action will be taken, it may be stochastic

- *Q-value*

The expected reward of action a

$$Q(a) := \mathbb{E}[R | A = a] = \sum_r r P(r | A = a)$$

- *Optimal Value*

Maximum expected reward

$$V^* := Q(a^*) = \max_{a \in \mathcal{A}} Q(a)$$

Multi-Armed Bandit: evaluating strategies

■ *Total Expected Regret*

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

For just one sequence of T trials, the *Total Regret with expected rewards* is

$$L(T) := TV^* - \sum_{t=1}^T Q(\pi(t))$$

action taken at step t

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

$$\bar{L}(T) := TV^* - \sum_{a=1}^N \mathbb{E}[T_a(T)]Q(a) = \sum_{a=1}^N \mathbb{E}[T_a(T)]\Delta_a$$

number of times action a is taken in T trials (i.e. a random variable)

where

$$\Delta_a := V^* - Q(a)$$

Multi-Armed Bandit: evaluating strategies

■ Total Expected Regret

$$\bar{L}(T) := TV^* - \sum_{a=1}^N \mathbb{E}[T_a(T)]Q(a) = \sum_{a=1}^N \mathbb{E}[T_a(T)]\Delta_a$$

number of times action a is taken in T trials (i.e. a random variable)

where

$$\Delta_a := V^* - Q(a)$$

With the optimal policy π^* the total expected regret is 0.

Whereas, with the *random policy* the total expected regret grows linearly over time:

$$\bar{L}(T) = \frac{T}{N} \sum_{a=1}^N \Delta_a \quad \dots \text{since, with a random strategy } \mathbb{E}[T_a(T)] = \frac{T}{N}$$

Multi-Armed Bandit: *Online learning*

Adaptive policy: *exploration vs. exploitation*

exploration: make trials over the set of N arms to improve on estimates $\hat{Q}(a)$

exploitation: make use of the current best estimates $\hat{Q}(a)$

■ Greedy policy

Initialize all the estimates $\hat{Q}(a)$ at random

Repeat:

- 1) select the bandit with the current best estimated reward $a = \operatorname{argmax}_a \hat{Q}(a)$
- 2) update the current estimate about a as

$$\hat{Q}(a) := \frac{\sum_{t=1}^{T_a} r_{a,t}}{T_a}$$

reward of arm a at trial t

Total number of times the arm a has been played

Multi-Armed Bandit: *Online learning*

Adaptive policy: *exploration vs. exploitation*

exploration: make trials over the set of N arms to improve on estimates $\hat{Q}(a)$

exploitation: make use of the current best estimates $\hat{Q}(a)$

- ϵ -greedy policy ($0 < \epsilon < 1$)

Initialize all the estimates $\hat{Q}(a)$ at random

Repeat:

- 1) with probability $(1 - \epsilon)$ select the bandit $a = \operatorname{argmax}_a \hat{Q}(a)$
else (*i.e. with probability* ϵ) select one bandit at random
- 2) update the current estimate about a

$$\hat{Q}(a) := \frac{\sum_{t=1}^{T_a} r_{a,t}}{T_a}$$

reward of arm a at trial t

total number of times the arm a has been played

Multi-Armed Bandit: *Online learning*

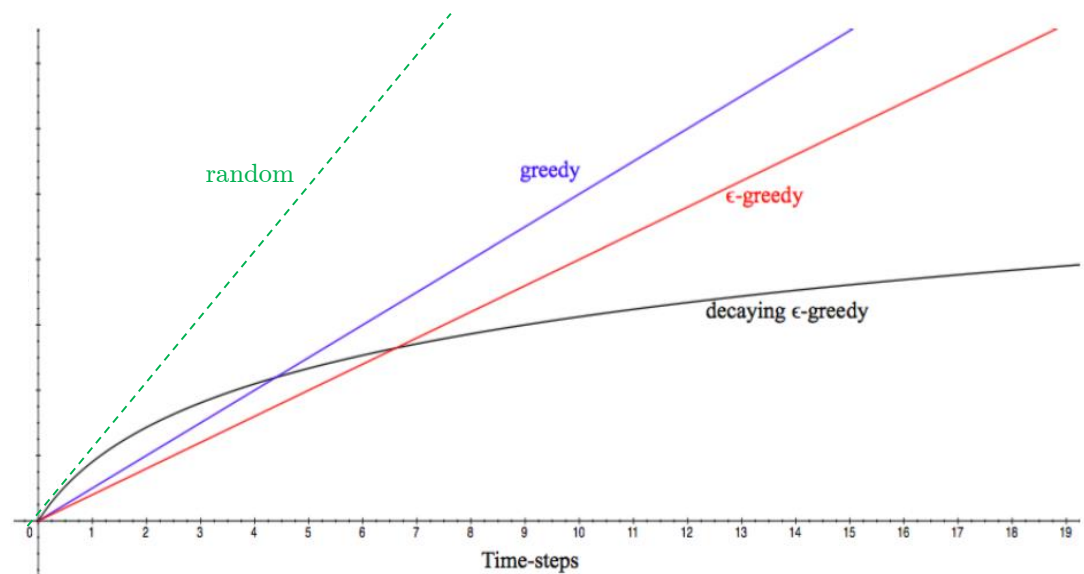
Adaptive policy: *exploration vs. exploitation*

exploration: make trials over the set of N arms to improve on estimates $\hat{Q}(a)$

exploitation: make use of the current best estimates $\hat{Q}(a)$

- Experimental comparison of different strategies (*Total Expected Regret*)

After a certain period of time, the *greedy* strategy stops exploring and exploits its estimates whereas, the ϵ -greedy strategy keeps exploring and improving



Decaying ϵ -greedy strategy: $\epsilon = \frac{\epsilon_{initial}}{t}$

Multi-Armed Bandit: evaluating strategies

- The two *greedy* strategies

They are *biased*: they depend on the initial random estimates

Optimistic variant: initially, set all $\hat{Q}(a) := 1$

The average total regret 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?

The decaying ε -greedy strategy does that...
Is there a minimum, i.e. a lower bound?

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 (i.e. $r \in \{0, 1\}$)

$$\lim_{T \rightarrow \infty} \bar{L}(T) \geq \ln T \sum_{a | \Delta_a > 0} \frac{\Delta_a}{\text{kl}(Q(a), V^*)} \quad \Delta_a := V^* - Q(a)$$

where

$$\text{kl}(Q(a), V^*) := Q(a) \ln \frac{Q(a)}{V^*} + (1 - Q(a)) \ln \frac{(1 - Q(a))}{(1 - V^*)}$$

\ the Kullback-Leibler divergence

In other words, we can achieve logarithmic growth for the total expected regret, but not better: on average, any adaptive strategy will choose suboptimal bandits a minimum number of times

$$\lim_{T \rightarrow \infty} \mathbb{E}[T_a(T)] \geq \frac{\ln T}{\text{kl}(Q(a), V^*)}$$

Multi-Armed Bandit: UCB strategy

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

Initialize all the estimates of the expected reward $\hat{Q}(a) := 0$

Play each arm once (to avoid zeroes in the formula below)

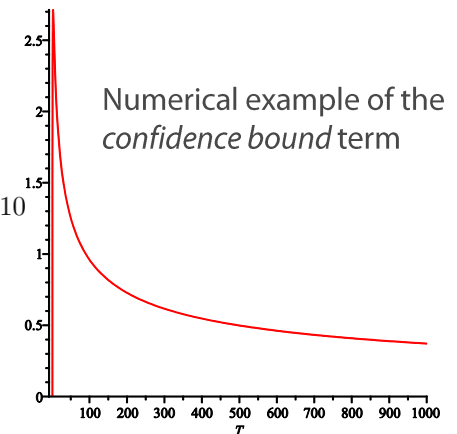
Repeat:

- select the bandit $a = \operatorname{argmax}_k \left(\hat{Q}(a) + \sqrt{\frac{2 \ln T}{T_a}} \right)$
- update the current estimate $\hat{Q}(a)$ as the *average* reward

total number of trials

number of times
the arm k has been played

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



Theorem

With the UCB strategy, $\lim_{T \rightarrow \infty} \mathbb{E}[T_a(T)] \leq \frac{8 \ln T}{\Delta_a^2} + c$

where it can be shown that $\frac{8}{\Delta_a^2} \geq \frac{1}{\operatorname{kl}(Q(a), V^*)}$

i.e. a (small) constant

(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 $\hat{Q}(a) \sim \text{Beta}(x; 1, 1)$

i.e. assume this as a random variable
with this distribution

Repeat:

- 1) sample each of the N distributions to obtain an estimate $\hat{Q}(a)$
- 2) select the bandit $a = \operatorname{argmax}_a \hat{Q}(a)$
- 3) update the *posterior* distribution

$$\hat{Q}(a) \sim \text{Beta}(x; R_a + 1, T_a - R_a + 1)$$

total number of times the arm has been played
total (*Bernoulli*) reward from this arm (*i.e. number of wins*)

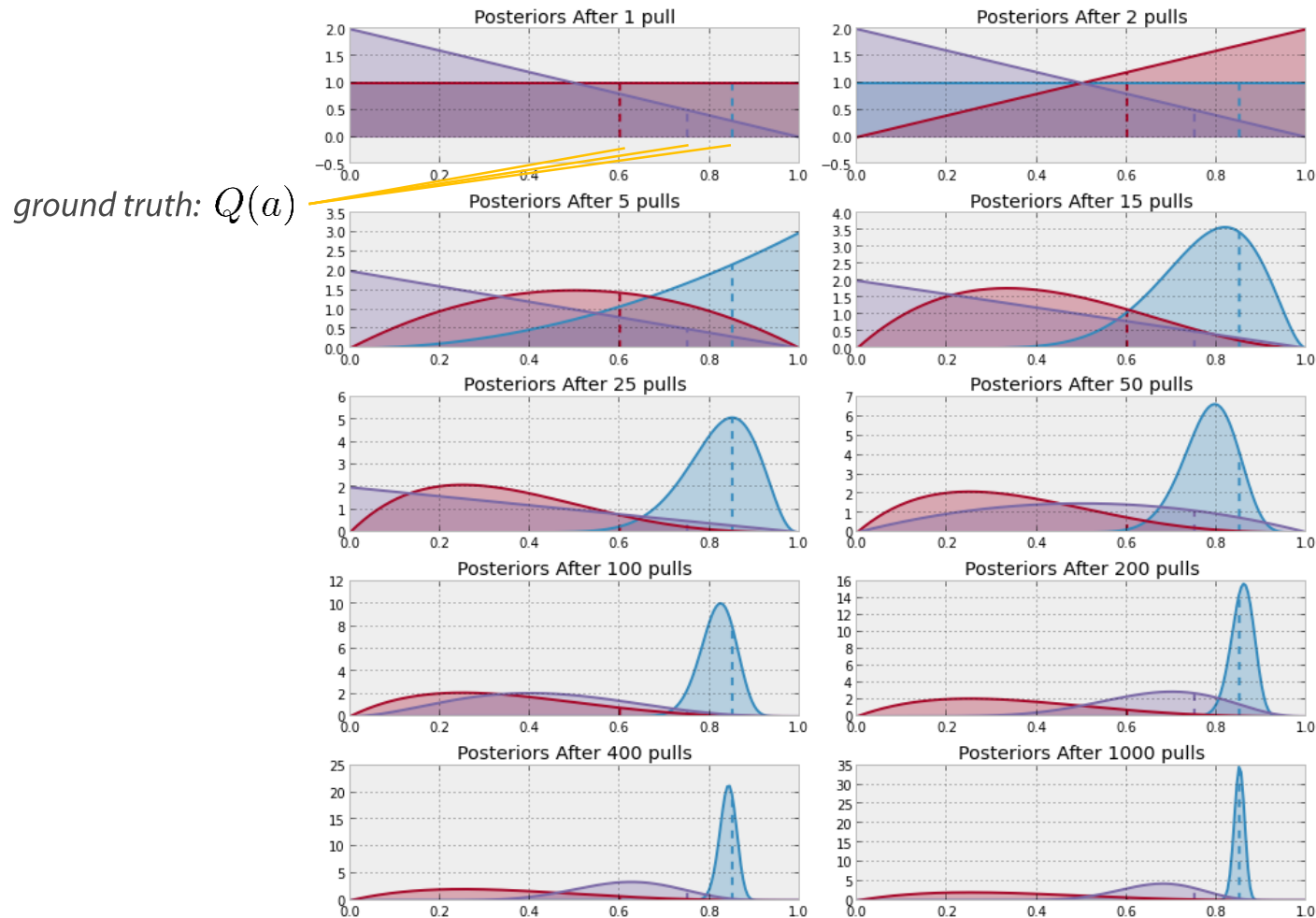
Theorem [Kaufmann et al., 2012]

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]

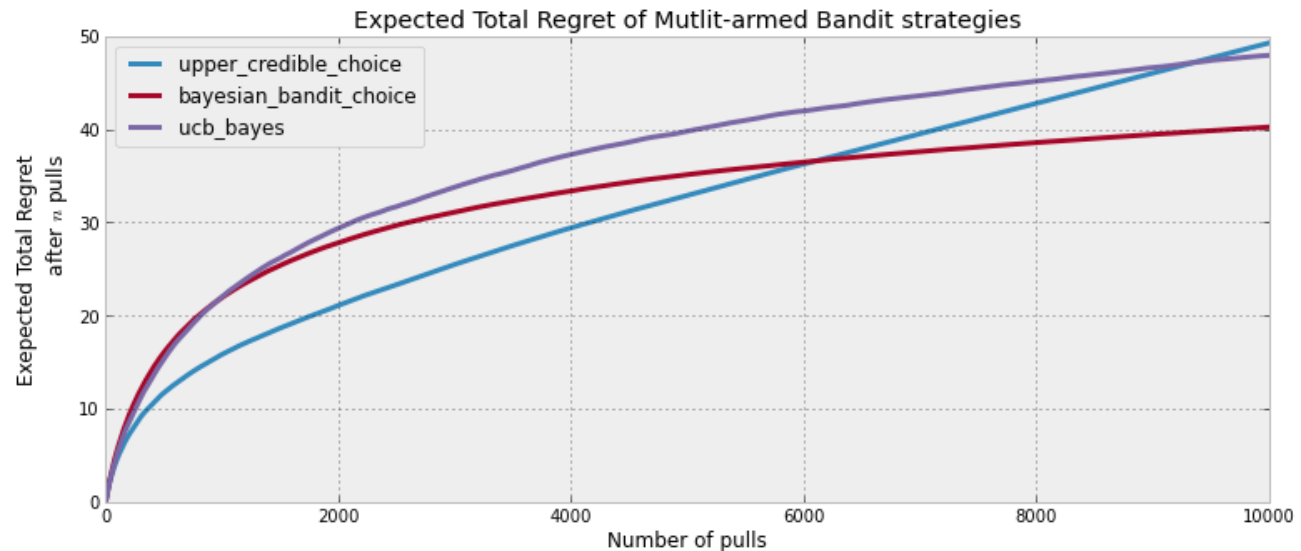
Example run with 3 arms: trace of the posterior probabilities for each $\hat{Q}(a)$



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]



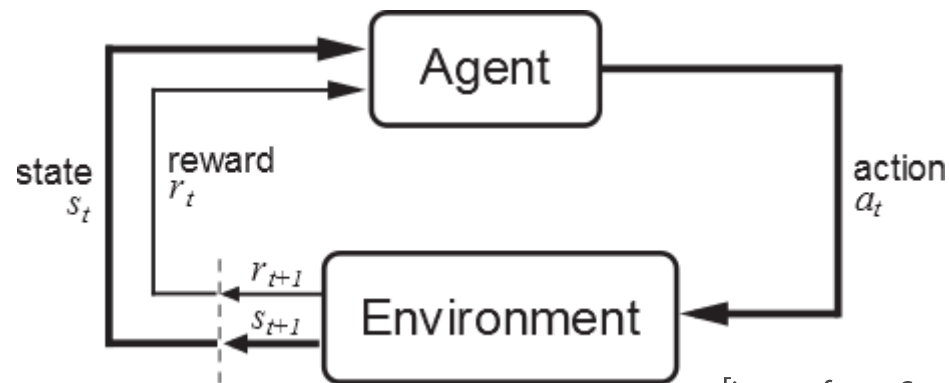
*Actually, Thompson Sampling is a preferred strategy at Google Inc.
(see <https://support.google.com/analytics/answer/2846882?hl=en>)*

[image from: <http://camdp.com/blogs/multi-armed-bandits>]

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?



[image from: Sutton, Barto, *Reinforcement Learning*. 1998]

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

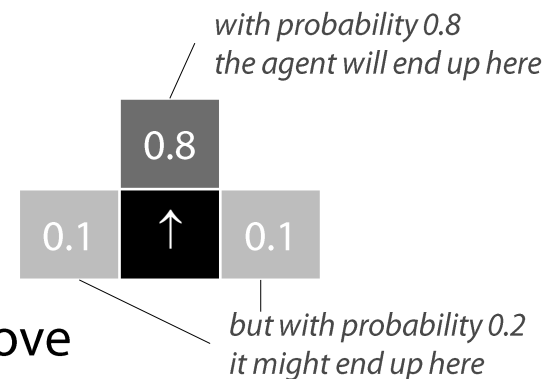
The *state* of the agent is the position on the grid:
e.g. (1,1), (3,4), (2,3)

At each time step, the agent can move one box
in the directions $\leftarrow \uparrow \downarrow \rightarrow$

*The effect of each move is somewhat stochastic, however:
for example, a move \uparrow 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)

	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: $\langle \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$

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

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

A reward function: $r : \mathcal{S} \rightarrow \mathbb{R}$

A transition probability distribution: $P(S_{t+1} | S_t, A_t)$ (also called a model)

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

$$P(S_{t+1} | S_t, A_t) = P(S_{t+1} | S_t, A_t, S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, \dots)$$

A discount factor: $0 \leq \gamma < 1$

Markov Decision Process (MDP): policies and values

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

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

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

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

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards could be 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

Note also that when the reward is *bounded*, i.e. $r(S) \leq r_{\max}$

$$\sum_{t=0}^{\infty} \gamma^t r(S_t) \leq r_{\max} \sum_{t=0}^{\infty} \gamma^t = r_{\max} \frac{1}{1-\gamma}$$

for $\gamma < 1$ this is the *geometric series*

Markov Decision Process (MDP): policies and values

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

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

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

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

Note the role of the *discount factor*: a value $\gamma < 1$ means that that future rewards could be 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 finite
Sooner or later the agent will fall into one of the absorbing states

Bellman equations

By working on the definition of value function:

$$\begin{aligned} V^\pi(s) &:= \mathbb{E}[r(S_t) + \gamma r(S_{t+1}) + \gamma^2 r(S_{t+2}) + \dots \mid \pi, S_t = s] \\ &= \mathbb{E}[r(S_t) + \gamma(r(S_{t+1}) + \gamma r(S_{t+2}) + \dots) \mid \pi, S_t = s] \\ &= r(s) + \gamma \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \dots \mid \pi, S_t = s] \\ &= r(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \dots \mid \pi, S_{t+1} = s'] \\ &= r(s) + \gamma \sum_{S_{t+1}} P(S_{t+1} \mid s, \pi(s)) \cdot V^\pi(S_{t+1}) \end{aligned}$$

This means that in a Markov Decision Process:

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

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

If the set of states is finite, there are exactly $|S|$ (linear) Bellman equations for $|S|$ variables: in general, for any deterministic policy, V^π can be computed analytically

Optimal policy – Optimal value function

- Basic definitions

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

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

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

With Bellman Equations:

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

$$\begin{aligned} V^*(s) &= r(s) + \gamma \max_{\pi} \left(\sum_{S_{t+1}} P(S_{t+1} | s, \pi(s)) \cdot V^*(S_{t+1}) \right) \\ &= r(s) + \gamma \max_a \left(\sum_{S_{t+1}} P(S_{t+1} | s, a) \cdot V^*(S_{t+1}) \right) \end{aligned}$$

Therefore:

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

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

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

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

Optimal value function: value iteration

Value iteration algorithm

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

Repeat:

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

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

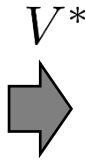
Theorem: for every fair way (i.e. giving an equal chance) of visiting the states in S , this algorithm converges to V^*

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



	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^*



Define the optimal policy as:

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

	1	2	3	4
1	→	→	→	1
2	↑		↑	-1
3	↑	←	←	←

π^*

Optimal policy: policy iteration

Policy iteration algorithm

Initialize $\pi(s), \forall s \in \mathcal{S}$ at random

Repeat:

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

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

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

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

It is also greedy, 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 action space:
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 learned is the optimal policy π^*

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

- Different conditions: *learning 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...

Action value function

An analogous of the value function V^π

Given a policy π , the **action value function** is defined, for each pair (s, a) as:

$$\begin{aligned} Q^\pi(s, a) &:= \sum_{S_{t+1}} P(S_{t+1} | s, a) \cdot V^\pi(S_{t+1}) \\ &= \sum_{S_{t+1}} P(S_{t+1} | s, a) \cdot \mathbb{E}[r(S_{t+1}) + \gamma r(S_{t+2}) + \dots | \pi, S_{t+1}] \\ &= \sum_{S_{t+1}} P(S_{t+1} | s, a) \cdot [r(S_{t+1}) + \mathbb{E}[\gamma r(S_{t+2}) + \dots | \pi, S_{t+1}]] \\ &= \sum_{S_{t+1}} P(S_{t+1} | s, a) \cdot [r(S_{t+1}) + \gamma Q^\pi(S_{t+1}, \pi(S_{t+1}))] \end{aligned}$$

In other words, $Q^\pi(s, a)$ is the expected value of the reward in S_{t+1} by taking action a in state s and then following policy π from that point on

Following a similar line of reasoning, 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

■ Q-learning algorithm (ϵ -greedy version)

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

Repeat:

- 1) Select the action $\operatorname{argmax}_a \hat{Q}(s, a)$ with probability $(1 - \epsilon)$
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

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

Exponential Moving Average
(see later...)

Note that step 1) is closely similar to a **multi-armed bandit**:
in each state, the agent has to choose one among all actions in \mathcal{A}
and this will produce a random reward...

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 $\alpha \rightarrow 0$ as experience progresses, then

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

with probability 1

*The Q-learning algorithm bypasses the MDP entirely,
in the sense that the optimal strategy is learnt without learning the model $P(S_{t+1} \mid S_t, A_t)$*

An aside: *moving averages*

Following non-stationary phenomena

■ Average

Definition:
$$\bar{v}_T := \frac{1}{T} \sum_{k=1}^T v_k$$

Running implementation:

$$\begin{aligned}\bar{v}_T &= \frac{1}{T} \left(v_T + \sum_{k=1}^{T-1} v_k \right) = \frac{1}{T} \left(v_T + (T-1)\bar{v}_{T-1} \right) \\ &= \bar{v}_{T-1} + \frac{1}{T} (v_T - \bar{v}_{T-1}) = \frac{1}{T} v_T + \left(1 - \frac{1}{T} \right) \bar{v}_{T-1}\end{aligned}$$

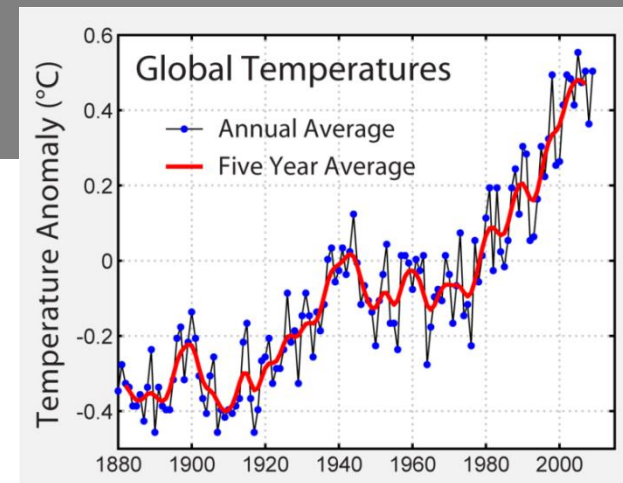
■ Simple Moving Average (SMA)

$$\bar{v}_{T,n} := \frac{1}{n} \sum_{k=T-n}^T v_k$$

■ Exponential Moving Average (EMA)

$$\bar{v}_{T,\alpha} := \alpha v_T + (1 - \alpha) \bar{v}_{T-1,\alpha}, \quad \alpha \in [0, 1]$$

“the weight of newer observations remains constant”



[image from wikipedia]

“the weight of newer observations diminishes with time”

An aside: moving averages

■ Exponential Moving Average (EMA)

$$\bar{v}_{T,\alpha} := \alpha v_T + (1 - \alpha) \bar{v}_{T-1,\alpha}, \quad \alpha \in [0, 1]$$

Expanding:

$$\begin{aligned} \bar{v}_{t,\alpha} &= \alpha v_t + (1 - \alpha) \bar{v}_{t-1,\alpha} \\ &= \alpha v_t + (1 - \alpha)(\alpha v_{t-1} + (1 - \alpha) \bar{v}_{t-2,\alpha}) \\ &= \alpha v_t + (1 - \alpha)(\alpha v_{t-1} + (1 - \alpha)(\alpha v_{t-2} + (1 - \alpha) \bar{v}_{t-3,\alpha})) \\ &= \alpha (v_t + (1 - \alpha) v_{t-1} + (1 - \alpha)^2 v_{t-2}) + (1 - \alpha)^3 \bar{v}_{t-3,\alpha} \end{aligned}$$

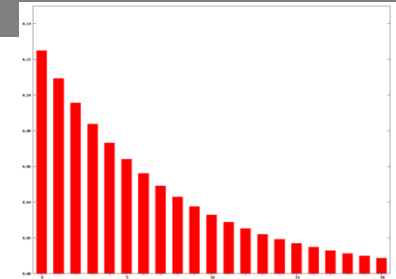
The weight of past contributions decays as

$$(1 - \alpha)^{\Delta t}$$

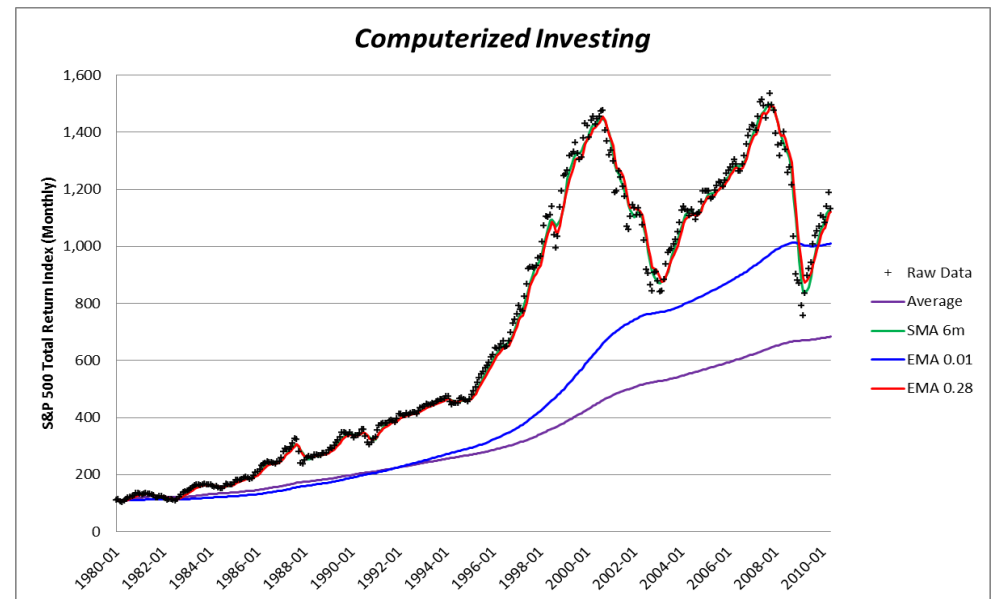
A SMA with n previous values is approximately equal to an EMA with

$$\alpha = \frac{2}{n + 1}$$

$(1 - \alpha)^{\Delta t}$
"the weight
of older observations
diminishes with time"



[image from wikipedia]



Q-Learning revisited

■ Q-learning algorithm (ε -greedy version)

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

Repeat:

- 1) Select the action $a = \operatorname{argmax}_a \hat{Q}(s, a)$ 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

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

By rewriting step 3)

$$\begin{aligned} \hat{Q}(s, a) &= \hat{Q}(s, a) + \Delta \hat{Q}(s, a) = \hat{Q}(s, a) + \alpha [r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)] \\ &= \alpha [r + \gamma \max_{a'} \hat{Q}(s', a')] + (1 - \alpha) \hat{Q}(s, a) \end{aligned}$$

Exponential Moving Average

compare with (see before):

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

Expectation

SARSA

▪ SARSA algorithm (ϵ -greedy version)

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

Repeat:

- 1) Select the action $a = \operatorname{argmax}_a \hat{Q}(s, a)$ with probability $(1 - \epsilon)$ otherwise, select a at random
- 2) The agent is now in state s' and has received the reward r
- 3) Select the action $a' = \operatorname{argmax}_a \hat{Q}(s', a)$ with probability $(1 - \epsilon)$ otherwise, select a' at random
- 4) Update $\hat{Q}(s, a)$ by

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

————— No more 'max' here

Q-learning is an *off-policy* algorithm: each update involves $\max_{a'} \hat{Q}(s', a')$
(i.e. *exploration* is not taken into account)

SARSA is an *on-policy* algorithm: each update involves $\hat{Q}(s', a')$
(which involves the next policy action, *exploration* included)

SARSA vs Q-Learning

Cliff World

'S' is the start

'G' is the goal

Each white box has $r = -1$

'The Cliff' region has $r = -100$
and entails going back to 'S'

Experimental Results

SARSA finds a sub-optimal but safer path
since its learning takes into account
the ϵ risk of going off the cliff

Q-learning finds the optimal path
but, occasionally, it falls off the cliff
during learning due to the ϵ -greedy strategy

