Learning Agents

Boi Faltings and Panayiotis Danassis

Laboratoire d'Intelligence Artificielle
boi.faltings@epfl.ch
http://moodle.epfl.ch/

Fall 2018

Machine learning for agents

Often, effects of actions are not known a priori:

- self-driving car: how steering/acceleration/braking affects trajectory.
- financial markets: market reaction.
- recommendation: does the user like this item?
- ad placement: select one of possible ads to display.

⇒ agent has to learn them from observation.

Learning to act

Different from learning from data:

- gathering data requires *exploration* which may be costly or even dangerous (e.g. medical trials).
- actions may have to be part of a sequence of steps to be beneficial.
- world is often not static and model has to adapt.
- other agents may change their behavior and invalidate the model.

Different form of learning: *reinforcement learning*.

Motivating Application - Medical Trial

- Treatment effectiveness against a new strain of a disease.
- Actions: $drug_1, drug_2, \ldots, drug_n$
- Reward:
  - 1 / 0 for success or failure,
  - Patient's condition improvement (%)
  - 1 / time to heal, etc.

- Goal:
  - Correctly identify the best treatment (exploration).
  - Treat patients as effectively as possible (exploitation)
- Trials:
  - expensive (find qualified patients, cost of drugs, etc.)
  - dangerous (patients may die or be ill)

$\Rightarrow$ minimize needs for trials.

Model for learning agents

- Agents maintain a *Q-table* that estimates the expected cumulative future reward of each action:

$$Q(s,a) \simeq r(s,a) + \sum_{s}^{\prime} Pr(s',s,a) \max_a Q(s',a)$$

- Q-table is learned from the agent observing effects of its actions.
- optimal action $a^* = argmax_a Q(s,a)$.
- Simpler model than MDP: details of state transitions not needed.

First consider model without state: learn $Q(a) = r(a)$.

Updating the Q-table

- Bayesian update: given observation $r(a_t)$ (e.g. 1 / time to heal):

$$Q_{t+1}(a) = \begin{cases} \alpha r(a) + (1-\alpha) Q_t(a) & \text{for } a = a_t \\ Q_t(a) & \text{otherwise} \end{cases}$$

- Moving average: $\alpha = 1/n(a,t)$ where $n(a,t)$ = number of times a was taken up to time t.
- Larger α gives more importance to recent observations.
- Every action gets played a large number of times
  $\Rightarrow$ estimates become precise
  $\Rightarrow$ Q-table converges to correct values.

Exploration-Exploitation Tradeoff

- We don't want to do random moves infinitely often!
- Need a strategy so that we can balance the need for *exploration* with the desire for *exploitation* what was already learned!
- Goal is to minimize overall *regret*: difference between the payoff with the optimal policy vs. payoff with the current policy.
  - exploration: improve the model to reduce future regret.
  - exploitation: use the model to reduce current regret.

Objective: minimize regret

- Agent can only do as good as the environment lets it.
- Regret l = difference in reward between
  - action a^* taken by the best static policy.
  - action taken by the agent's policy π.
- Goal is to minimize *cumulative regret* L_T:

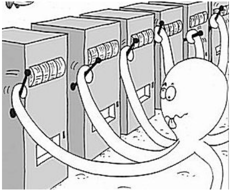
$$L_T = \sum_{t=1}^{T} l_t = max_a \sum_{t=1}^{T} r_t(a) - \sum_{t=1}^{T} r_t(\pi)$$

- Learning without state
  $\Rightarrow$ optimal action does not change over time
  $\Rightarrow$ compare with best single action a^* taken at every step.

The Multi-Armed Bandit (MAB) Problem

A slot machine (bandit) with multiple arms:

- Each arm (move) generates rewards with an unknown probability distribution.
- Rewards are only observed when playing the i^{th} arm.
- A player uses a policy to choose the next arm based on previous plays.

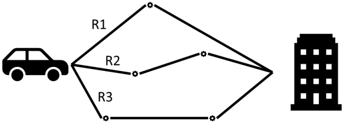


Objective

Maximize the sum of the rewards over many plays.

Example

- Reward for each action is independent and identically distributed.
- E.g. route selection:
  - $$Q_{t+1}(a) = \begin{cases} \alpha r(a) + (1-\alpha) Q_t(a) & \text{for } a = a_t \\ Q_t(a) & \text{otherwise} \end{cases}$$
  - Assume max travel time 20 min $\Rightarrow r(a) = 1 - \frac{time}{20}, \alpha = 0.1$



Time (min)

R1	-	-	12	14
R2	18	-	-	-
R3	-	9	-	-

➡

Q table

1	1	0.94	0.88	
0.91	0.91	0.91	0.91	...
1	0.96	0.96	0.96	

Example (2)

- E.g. slot machines:
  - $$Q_{t+1}(a) = \begin{cases} \alpha r(a) + (1-\alpha) Q_t(a) & \text{for } a = a_t \\ Q_t(a) & \text{otherwise} \end{cases}$$
  - Assume max prize 20CHF $\Rightarrow r(a) = \frac{prize}{20}, \alpha = 0.1$

Prize (CHF)					Q table				
-	10	-	11	15	1	0.95	0.95	0.91	0.894
9	-	20	-	-	0.945	0.945	0.951	0.951	0.951

Epsilon-greedy Strategy

- Choose best action according to current Q-table with probability $1 - \epsilon$, choose a random action otherwise.
- As $t \to \infty$, clearly every action is tried an infinite number of times.

$\Rightarrow$ Q-learning will converge to the true table.

$\Rightarrow$ eventually, the average regret when selecting the optimal action will approach 0.

Motivating Application - Medical Trial (2)

- ϵ-greedy exploration:
    - High probability: prescribe the most successful drug.
    - Periodically prescribe random ones.

Performance of ϵ-greedy

After convergence:

- With probability $1 - \epsilon$, agent takes optimal action
- However, always takes a suboptimal action with probability ϵ.

$\Rightarrow$ cumulative regret increases linearly with time ($O(c\epsilon t)$).

- Note: $c = max_{a,a'} r(a) - r(a')$: bigger differences between actions cause bigger regret.

Decreasing ϵ

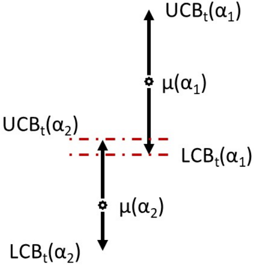
- Growth of regret depends on ϵ: decreasing ϵ with time can reduce regret growth!
- Example: $\epsilon \sim 1/t$: cumulative regret $L_t = \int_t 1/t = O(\log t)$.
- However, need to also ensure that Q-learning receives sufficient samples to converge.

$\Rightarrow$ rate would have to depend on knowing true action rewards.

- Need a way to estimate convergence.
- Major flaw: exploration schedule does not depend on history of rewards (non-adaptive exploration).

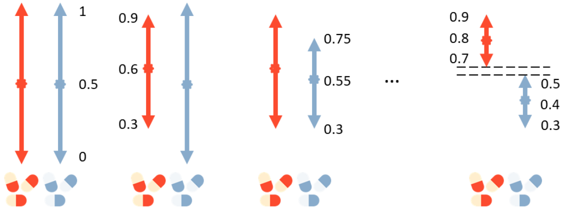
Confidence Bounds

- Maintain *confidence bounds* on the average rewards $\mu(\alpha)$ of each action α based on observations: $Pr[LCB(\alpha) < \mu(\alpha) < UCB(\alpha)] \geq 1 - \delta$
- Successive elimination: alternate α_1, α_2 until $UCB_T(\alpha_2) < LCB_T(\alpha_1)$ $\Rightarrow$ eliminate α_2 since optimal only with small probability $< 2\delta$.
- Optimism under uncertainty: pick the action with the best upper bound.

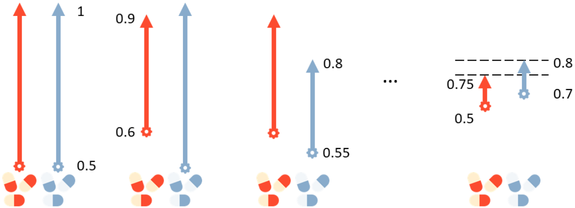


Motivating Application - Medical Trial (3)

Successive Elimination:



Upper confidence bound (optimistic):



Upper confidence bounds

- Hoeffding bound (for any random variable X, t samples):

$$Pr[X_{t+1} > \bar{X}_t + u] \leq e^{-2tu^2}$$

- Suppose we want $Pr[Q^* > Q_t + u] \leq \delta$, N_t samples:

$$u(a) = \sqrt{\frac{-\log \delta}{2N_t(a)}} = \sqrt{\frac{2\log t}{N_t(a)}}$$

  where second equality holds when choosing $\delta = t^{-4}$:

- (optimistic) UCB1 algorithm: assume actual reward close to upper bound; select best action according to $Q_t(a) + u(a)$.

$\Rightarrow$ actions with small N_t are played more often.

Bounding expected regret

- Theorem: UCB1 algorithm achieves expected cumulative regret:

$$lim_{t\to\infty} L_t \leq \frac{8 \log t}{\sum_a \Delta_a}$$

where *gap* Δ_a is the difference $max_{a'} r(a') - r(a)$.

- Logarithmic in t
- Very similar rewards $\Rightarrow$ learning takes longer to distinguish.
- No matter what the (fixed) distribution of rewards.
- Agent only needs to observe reward for the action it has actually taken.

Adversarial bandits

- What if rewards can change over time:
  - randomly?
  - set by an adversary to fool the learning algorithm?
    e.g. rewards: (0, 1), (1, 0), (0, 1), ...
- Focus on adversarial setting as worst case.
- Insight: play needs to be random, otherwise adversary can always make the chosen action have the worst payoff.

Motivating Examples

- Medical trial:
  - Same treatment to everyone $\Rightarrow$ antibiotic resistant bacteria $\Rightarrow$ low reward.
  - $\Rightarrow$ maintain a mixture of antibiotics.
- Financial investments:
  - Want to minimize worst-case loss
  - $\Rightarrow$ diversify into different investments.

Adversarial rewards

- Adversary knows policy $\pi(a) = Pr(a)$ and tries to maximize regret.
- Baseline policy: agent always plays action a^*
- Regret:

$$L_T = max_{a^* \in A} \sum_{t=1}^{T} r_t(a^*) - \sum_{a \in A} \pi(a) r_t(a)$$

- Deterministic ($\pi(a_t) = 1$): adversary sets $r(a_t) = 0, r(s_t, a^*) = 1$ for some a^* that is rarely chosen.
- if $\exists a$ that is never chosen, regret $L_T = t$.

Randomized play

- Randomized ($\pi(a_t) = 1/k$): adversary has no influence on expected reward, $= 1/k$ whatever the reward distribution.
- Adversary would choose a^* to maximize L_t.
- No good choice: all choices of a^* have the same regret $(k-1)/k$.
- However, regret is large: can we find better randomized strategies?

Adversarial strategies

- Stage regret of action a when agent played π_t:

$$l(a)_t = r(a)_t - r(\pi_t)_t$$

- Cumulative regret of action a:

$$L_T(a) = \sum_{t=1}^{T} l(a)_t$$

- Insight: whenever agent plays a, cumulative regret of a does not change.

$\Rightarrow$ Regret matching: play a with the largest positive cumulative regret to reduce its relative cumulative regret.

$\Rightarrow$ Gives adversary no good options to choose a^* to maximize regret!

Regret matching

- Assumption: can observe rewards that would be obtained for all actions (even if the action was not taken).
- Regret matching: play action a with probability proportional to its cumulative regret.
- Adversary can change rewards to hurt policy π; policy converges to the best response.

$\Rightarrow$ cumulative regret grows as $\sqrt{t}$.

- Adversarial setting: worse than stationary setting ($O(\log t)$).

Multiplicative weight update

- Policy = probability distribution P(a): action a is taken with probability P.

$$P(a) = \frac{w(a)}{\sum_a w(a)}$$

- Initialize weights as $w(a) = 1$.
- Update weight of each action according to its reward:

$$w(a)_{t+1} = w(a)_t(1 + \epsilon R_t(a)/B)$$

  then adjust probabilities through normalization ($B =$ upper bound on possible rewards).
- Bound on cumulative regret (assuming B=1):

$$\sum_{t=1}^{T} E_{P(t,a)} R_t(a) - E_{P^*(t,a)} R_t(a) \leq \frac{\log |A|}{\epsilon} + \epsilon T$$

Regret bounds

- Choose $\epsilon = 1/\sqrt{T} \Rightarrow \mathit{regret} = O(\log|A|\sqrt{T})$
- Weaker bound than for UCB ($O(\log T)$), but can handle adversarial rewards.
- Note: regret is against a single best policy played throughout all T steps (not varied by adversary).

Exponential weight update (exp-3)

- Learning from only the rewards of the action actually taken $\Rightarrow$ distribution of samples is unbalanced.
- Instead of multiplicative update, multiply weight $w(a)$ by an exponential:

$$w(a)_{t+1} = w(a)_t e^{\epsilon R_t(a)/p_t(a)}$$

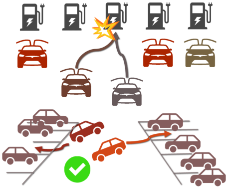
$\Rightarrow$ changes are stronger when action is not very likely, to even out convergence.

- best cumulative regret bound is $O(\sqrt{T|A|\log|A|})$ (when setting $\epsilon \propto 1/\sqrt{t}$).

Example

- Autonomous vehicles trying to acquire a charging station, or parking spot (two available resources: r_1, r_2).
- Repeated interactions until successful access.
- Binary reward 1 / 0 (success / failure).

- UCB1 *fails*:
  - Initially $UCB(r_1) = UCB(r_2)$.
  - Assume both initially select r_1 (collision) $\Rightarrow UCB(r_1) < UCB(r_2)$ for both agents.
  - Since $UCB(r_1) < UCB(r_2)$, both select r_2 (collision).
  - Next, both select r_1 (collision).
  - ...
- EXP3 *randomizes*!

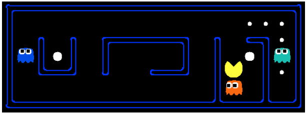


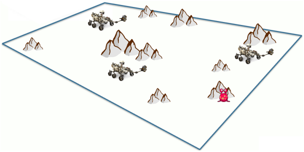
Learning with state

- Models so far learn to optimize instantaneous rewards.
- What if:
    - rewards also depend on state, and
    - actions also cause unknown state transitions?

Examples

- Video Games
  - Position of pac-man, ghosts, dots
  - Active power pellet?
  - Time, #lives
- Robotics
  - Position
  - Goal
  - Teammates
  - Obstacles





One bandit per state

- Straightforward solution: one bandit per state.
- However, requires collecting data for each state: convergence time multiplied by the number of states.
- Usually not feasible.

Contextual Bandits

- Define small set of *contexts* = groups of states with similar features. (e.g. medical trials ⇒ patient history, add placement ⇒ user profile, etc.)
- Instead of one bandit per state, have one bandit per context.

⇒ Learning requires less data, but quality depends on how well contexts fit the problem.

- Example: EXP4 algorithm.
- However, no consideration of state transitions!

Q-learning

- Learn table $Q(s, a)$ = sum of instantenous reward plus discounted value of successor state.
- start with arbitrary initial values.
- Observations:

  (s,a,r,s')

  action a in state s gives reward r and leads to state s'.
- $\Rightarrow$ improve current value of $Q(s, a)$ using *Q-learning rule*:

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha(r + \gamma max_{a'} Q(s', a'))$$

  α = learning factor $\in [0..1)$, must decrease over time
- Q-learning is guaranteed to converge to the optimum, as long as sample is representative.

Q ⇒ optimal policy

Optimal policy: take action that maximizes Q:

$$\pi(s) = \operatorname*{argmax}_a Q(s,a)$$

However, as before there is a tradeoff between *exploration* and *reward*.

Convergence of Q-learning

- Let
  - $Q^*(s,a)$ be the true maximum reward reachable from state s by taking action a.
  - $Q_t(s,a)$ the estimate at time t.
  - $\Delta_t(s,a) = max_{s,a}|Q_t(s,a) - Q^*(s,a)|$.

  Letting s' be the state following s, we have

$$
\begin{aligned}
\Delta_{t+1}(s,a) &= |\{R + \gamma max_{a'} Q_t(s',a')\} - \{R + \gamma max_{a''} Q^*(s',a'')|\} \\
&= \gamma |max_{a'} Q_t(s',a') - max_{a''} Q^*(s',a'')| \\
&\leq \gamma max_{a'''} |Q_t(s',a''') - Q^*(s',a''')| \\
&\leq \gamma max_{s'',a'''} |Q_t(s'',a''') - Q^*(s'',a''')| \\
&= \gamma \Delta_t
\end{aligned}
$$

Convergence of Q-learning (2)

- $\gamma < 1 \Rightarrow \Delta_{t+1} < \Delta_t$
- Thus, in the limit, the difference between $Q_t(s, a)$ and $Q^*(s, a)$ goes to zero!
- However, requires that all states and actions are visited sufficiently often.

Exploration-Exploitation tradeoff in Q-learning

- Convergence of Q-learning requires that all states and actions are visited a sufficient number of times.

$\Rightarrow$ same issue as in bandit problems.

- However, complicated by state transitions: not all states are easily reachable.
- Transitions cause dependencies between successive states that contradict the independence assumptions of bandit algorithms.
- Extreme case: transition graph is not connected $\Rightarrow$ complete exploration impossible.

Initializing the Q-table

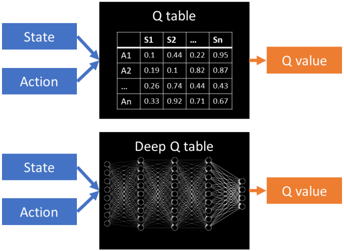
- Initialize all rewards to very high values $\Rightarrow$ action selection will pick actions that have not been tried a lot.
- However, space is very large $\Rightarrow$ will take a long time to converge.
- ϵ-greedy may be a good alternative.
- hard to prove optimality or even convergence: depends on state transitions.

Deep q-learning

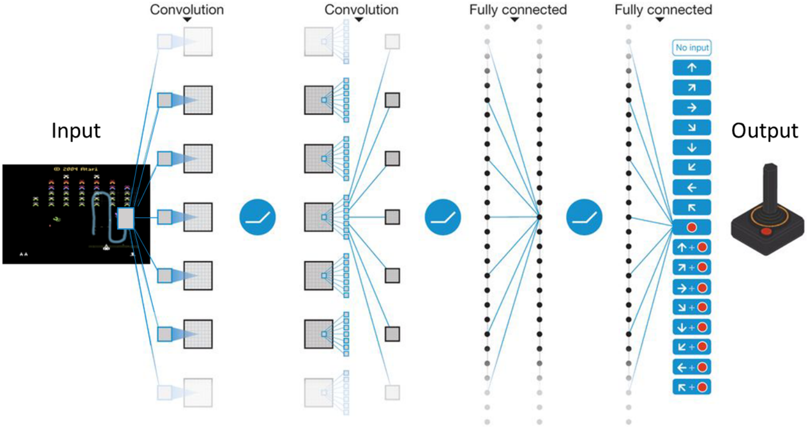
- Q-table can be very large, and take a lot of data to fill in.
- Assumption: entries in the Q-table have a regular distribution.

$\Rightarrow$ represent by a deep neural net.

- Advantage: generalization reduces need to try many similar actions.



Example


Input
Convolution
Convolution
Fully connected
Fully connected
No input
Output

Experience replay

- Sequences of observed states follow state transitions.

$\Rightarrow$ creates dependencies that destroy convergence of stochastic gradient descent.

- Avoid by rearranging observation tuples in random order.
- Developed by Google Deep Mind to play Atari videogames.

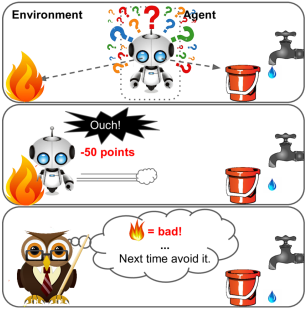
Most difficult example

Learning from a Teacher

- Key to efficient Q-learning: avoid trying bad actions.
- Observing a teacher shows only "good" transitions.

$\Rightarrow$ learning is much more efficient.

- However, agent will not become better than the teacher!



Summary

- Bandit problems: learn effect (reward) of each action.
- Exploration-Exploitation tradeoff.
- Fixed stochastic reward structure: greedy/UCB algorithms, cumulative regret $= O(\log T)$.
- Adversarial rewards: learn randomized strategy.
  - regret matching/multiplicative weight update
  - exponential weight update

  cumulative regret $= O(\sqrt{T})$.
- Learning with state
  - contextual bandits
  - Q-learning