20. Topics in Online Learning

- Random states
  - Bandits
  - ϵ-first
  - ϵ-greedy
  - Upper confidence bound
- Unobserved states
  - Robotics
  - Markov decision processes
  - Value / policy iteration
  - Linear programming implementation
  - Actor-Critic methods

This is a “for fun” lecture

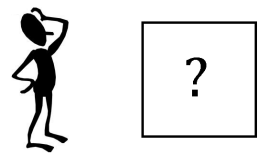
- Get some popcorn
- Put your feet up
- Enjoy!

The umbrella of online optimization

$$\underset{x}{\text{minimize}} \quad g(x)$$ where g depends on something unknown

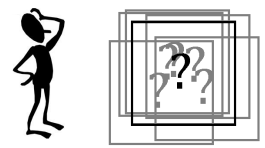
- **Unobserved state**: w is unknown

$$\underset{x}{\text{minimize}} \quad f(x; w)$$



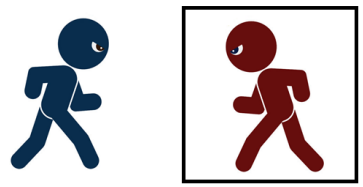
- **Stochastic state**: $w \sim \mathcal{D}$ is a random variable

$$\underset{x}{\text{minimize}} \quad \mathbb{E}_w[f(x; w)]$$



- **Adversarial state**: w is adversarial

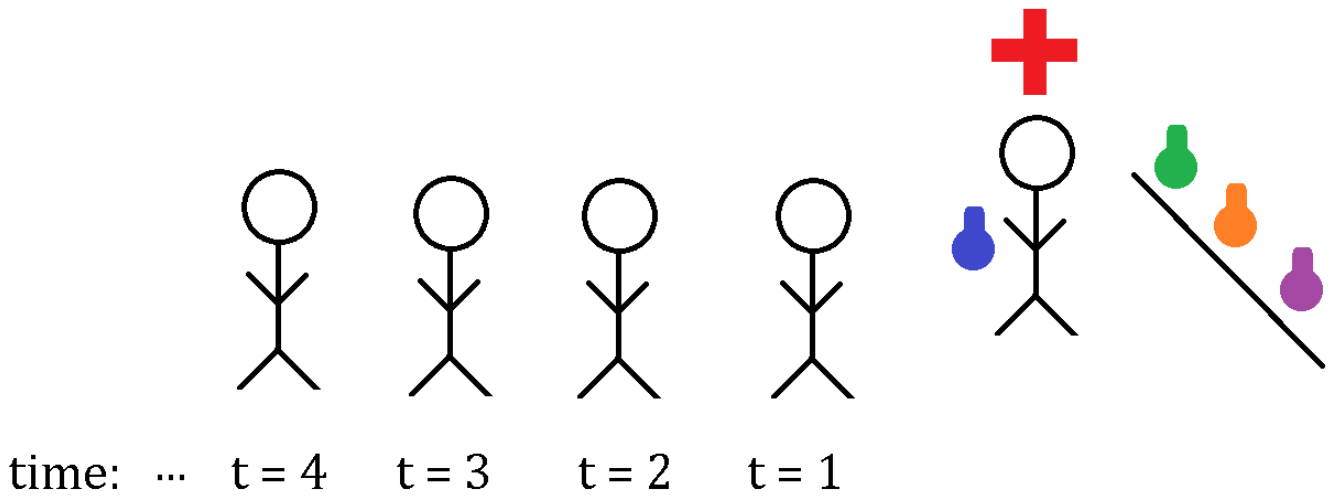
$$\underset{x}{\text{minimize}} \ \max_{w} f(x; w)$$



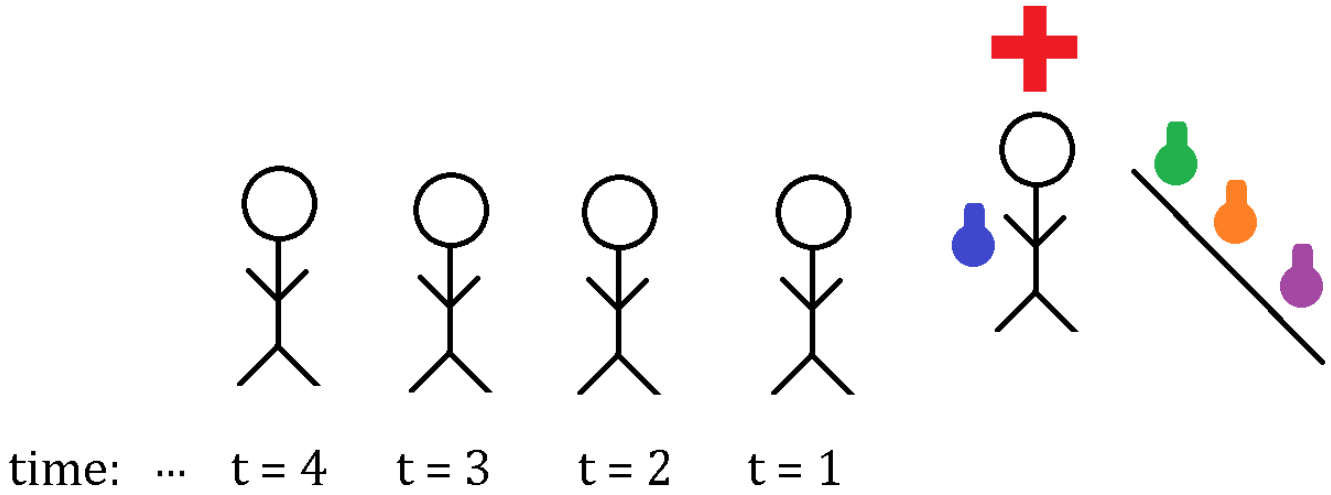
Stochastic online learning

A drug delivery problem

- m drugs, all super expensive, so only 1 drug per person
- Each person has a slightly different response to each drug
- Which drug should I recommend?



A drug delivery problem



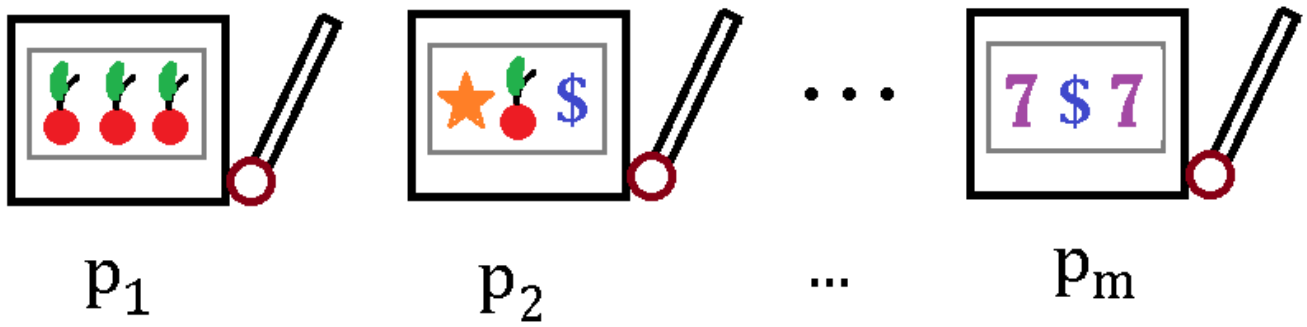
- Suppose each drug i has efficacy distributed as $\mathcal{N}(\mu_i, \sigma_i)$
- If I knew μ_i for $i = 1, ..., m$ then I just give drug k where

$$k = \underset{i}{\text{argmax}}\ \mu_i$$

- Best average efficacy

This is a (slightly harder) "bandit" problem

Bandit problem



- $p_i =$ probability that arm i will win
- There is a ground truth p_i, but you do not know it
- The ground truth never changes
- General idea: estimate

$$p_i \approx \bar{p}_i = \frac{\#\text{ times arm } i \text{ wins}}{\#\text{ times I pull arm } i}$$

- Better guess if I pull it a lot, but that wastes money!

Regret

$$\text{Regret}(T) = \sum_{t=1}^{T} \text{loss}(A_t) - \text{loss}(A^*)$$

where

- A_t is the algorithm at time t
- A^* is the best algorithm possible

Sublinear regret:

$$\lim_{T \to \infty} \frac{\text{Regret}(T)}{T} \to 0$$

ϵ-first method

- Total T trials
- For ϵT trials, explore
  - Pull arm i with uniform probability
  - Update probability

$$\bar{p}_i = \frac{\#\text{ times arm } i \text{ wins}}{\#\text{ times I pull arm } i}$$

- For $(1-\epsilon)T$ trials, exploit
  - Pull arm $i = \underset{i'}{\operatorname{argmax}}\, \bar{p}_{i'}$

- This also achieves sublinear regret!

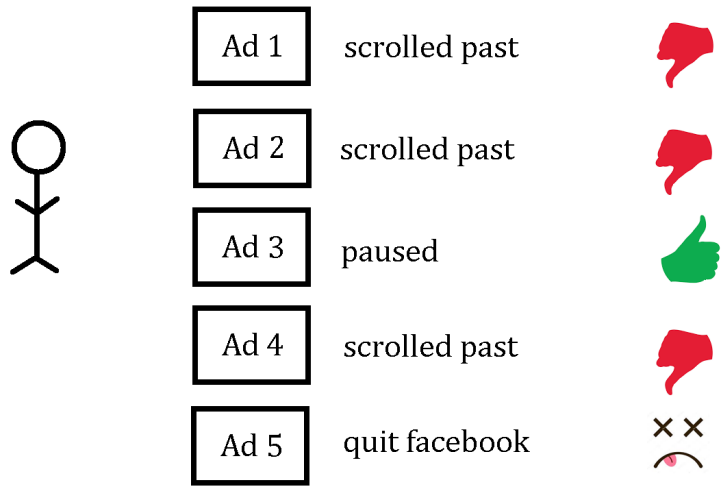
ϵ-greedy method

- With probability ϵ, explore
  - Pull arm i with probability $1/m$ (uniform sampling)
- With probability $1-\epsilon$, exploit
  - Pull arm $i = \underset{i'}{\operatorname{argmax}} \bar{p}_{i'}$
  - Break ties with uniform sampling
- Update probability

$$\bar{p}_i = \frac{\#\text{ times arm } i \text{ wins}}{\#\text{ times I pull arm } i}$$

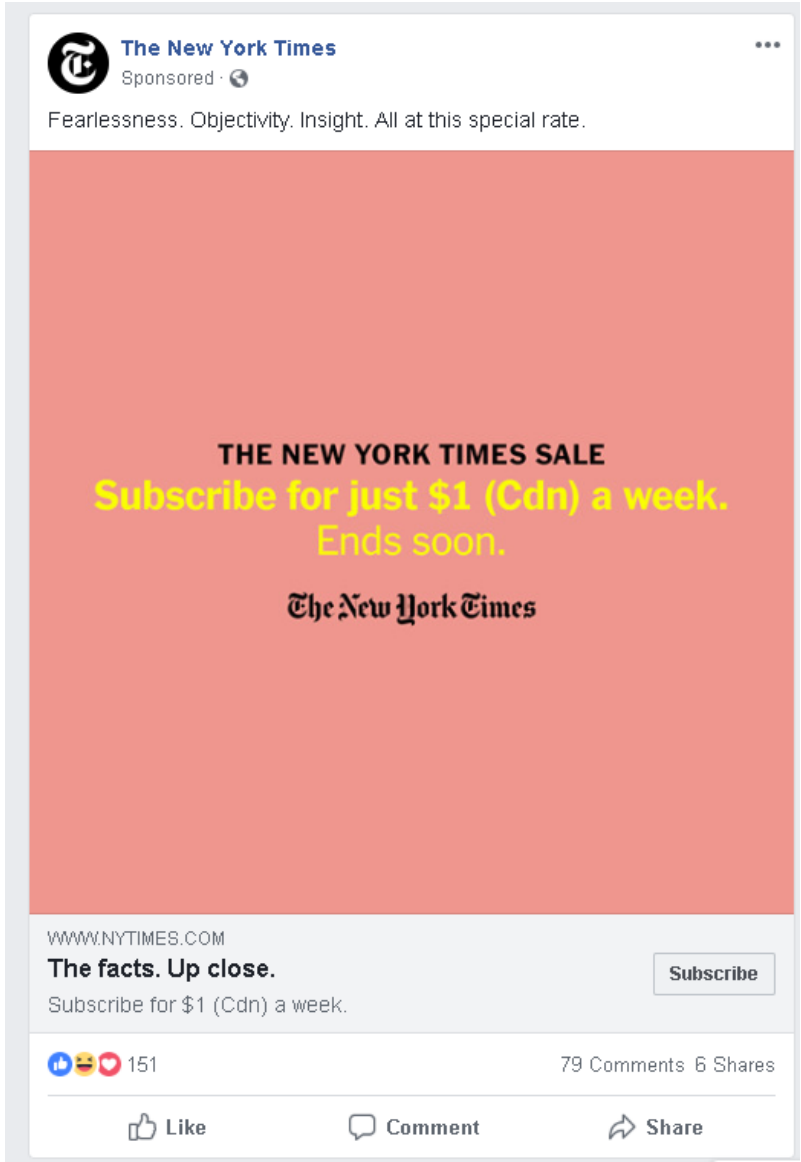
- This simple strategy achieves sublinear regret!

Ad placement

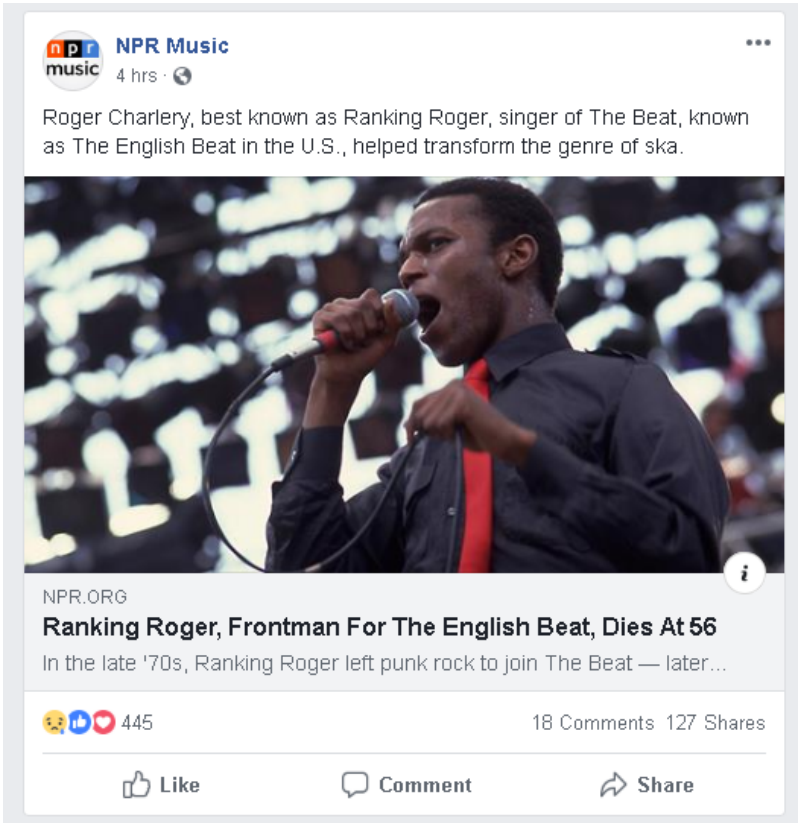


- You show Joe an ad
- If Joe clicks or pauses on it $\rightarrow$ positive feedback!
- If Joe scrolls past it $\rightarrow$ negative feedback
- Limited budget: too many bad ads and Joe will stop paying attention

My ad trajectory

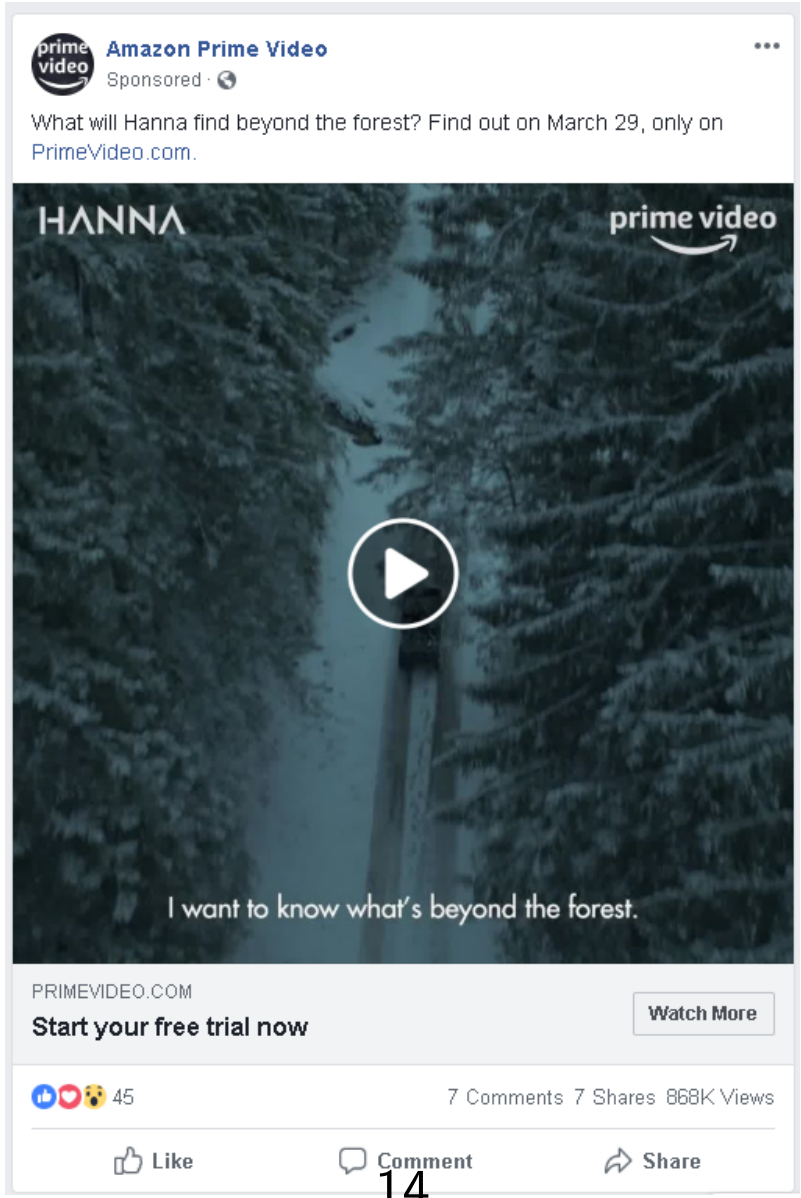


My ad trajectory

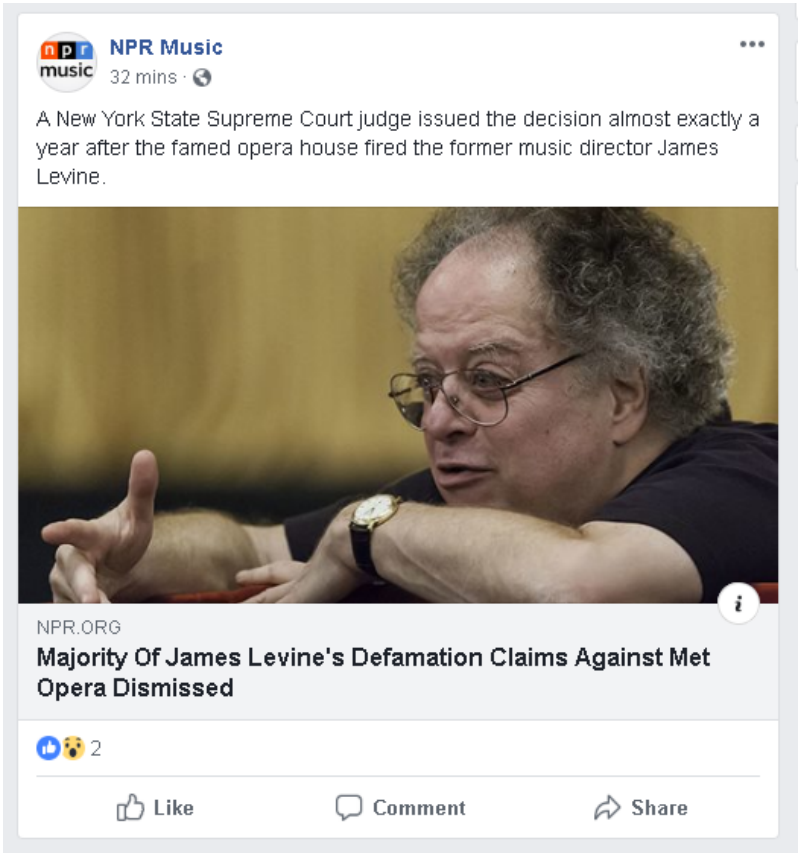


Explore

My ad trajectory

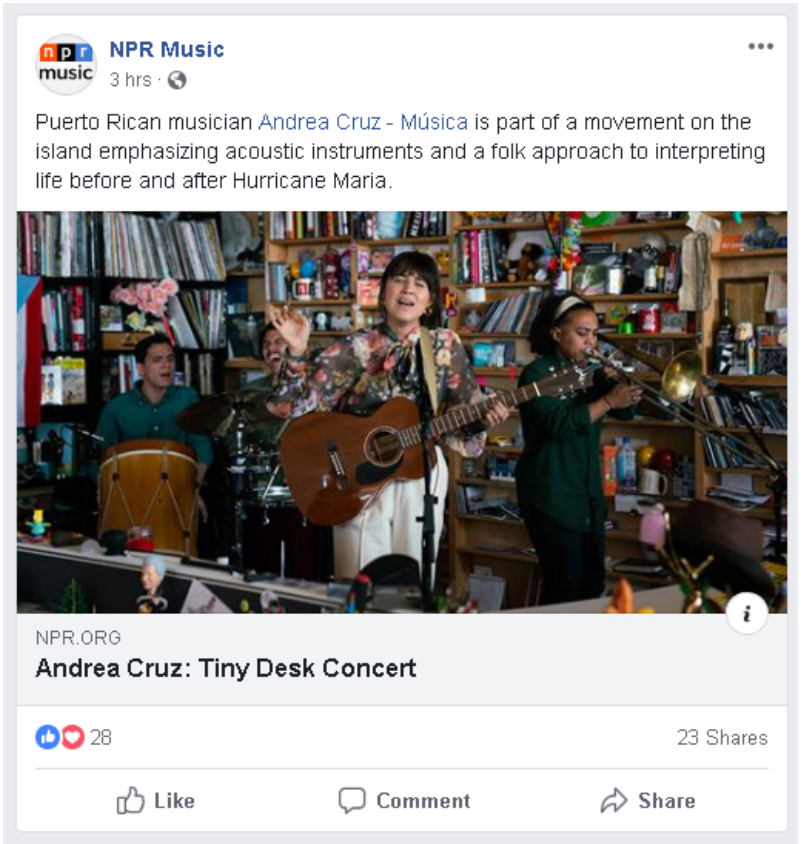


My ad trajectory



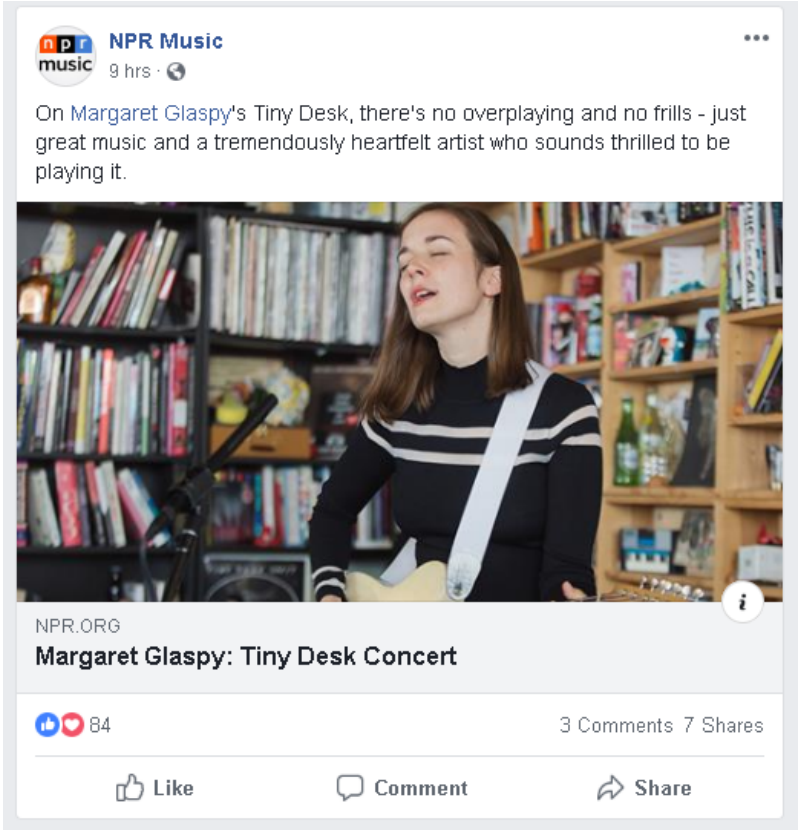
Explore

My ad trajectory



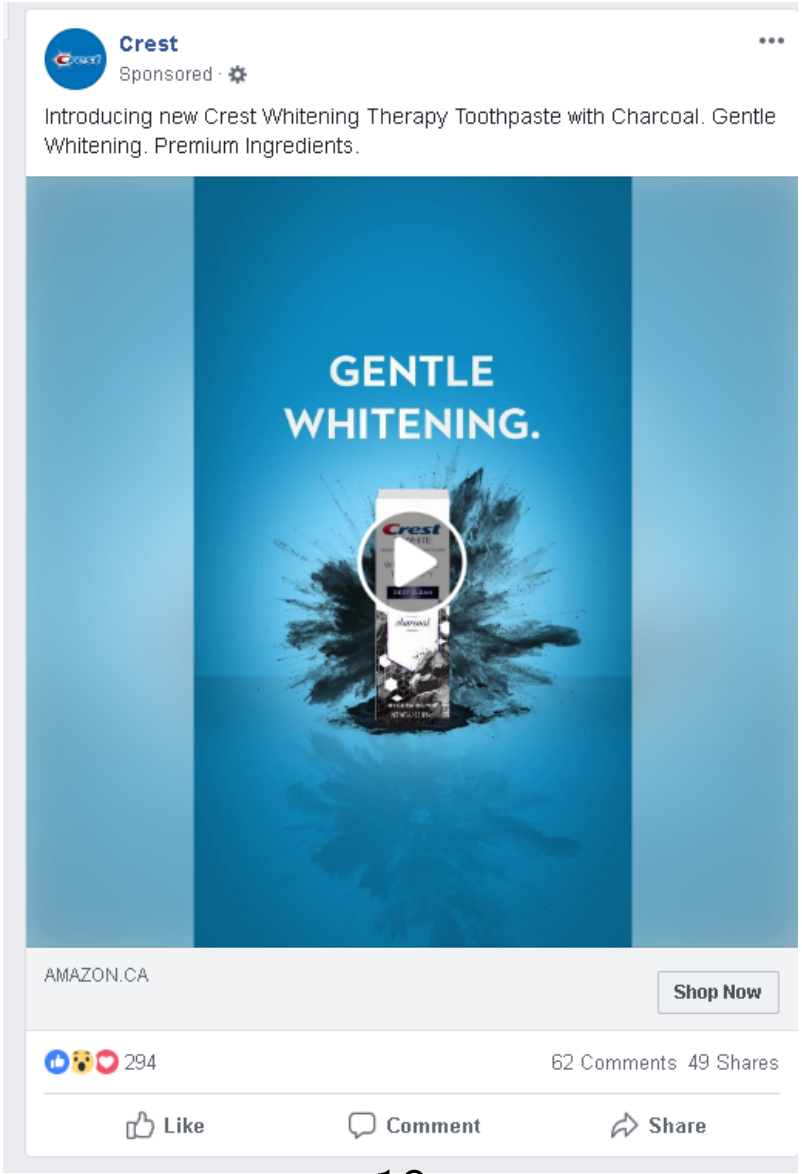
Exploit?

My ad trajectory

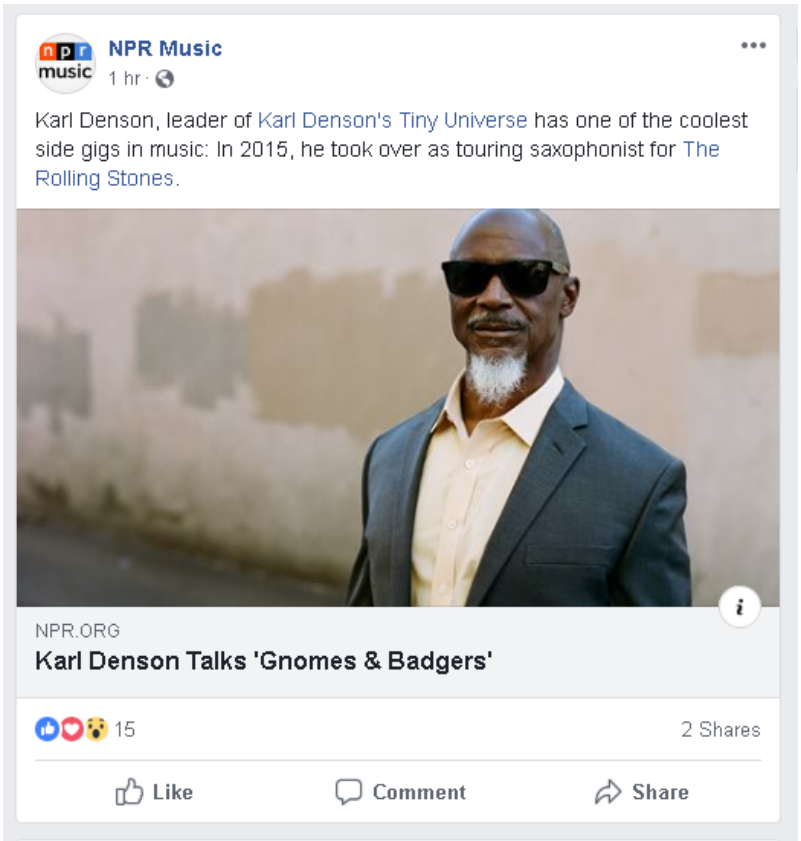


Exploit?

My ad trajectory



My ad trajectory

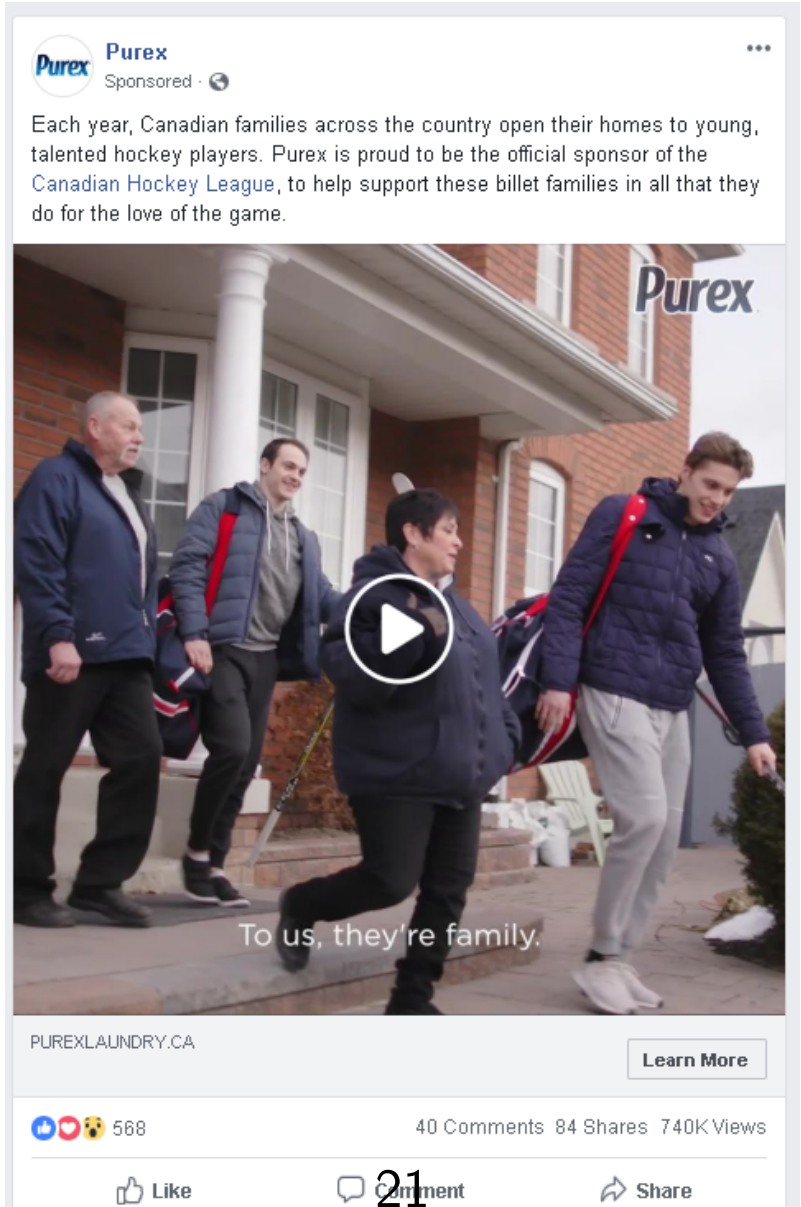


Exploit?

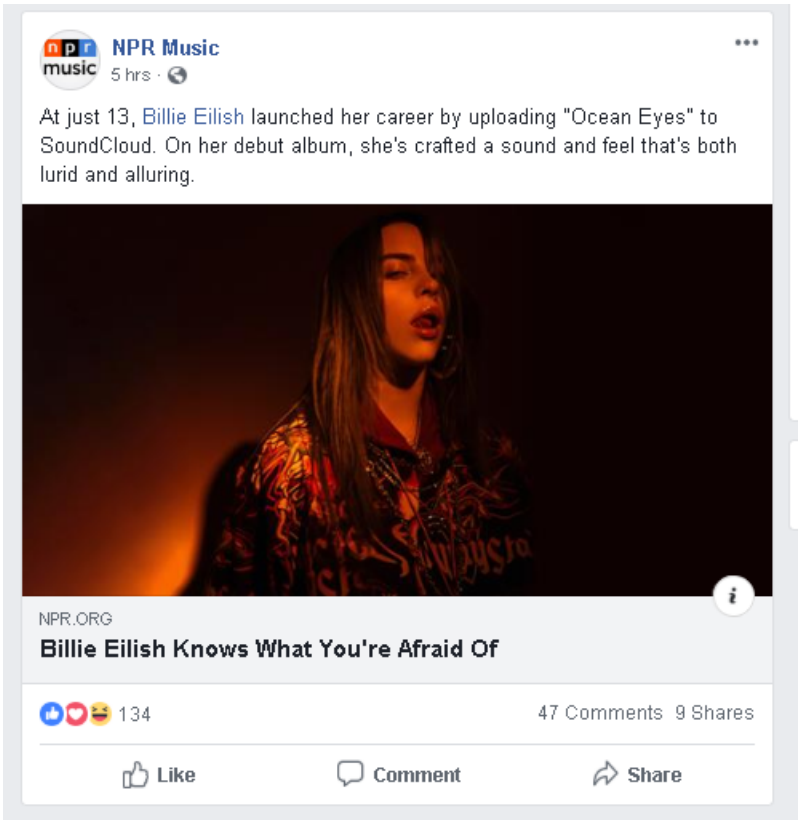
My ad trajectory

Explore (contextual?)

My ad trajectory



My ad trajectory

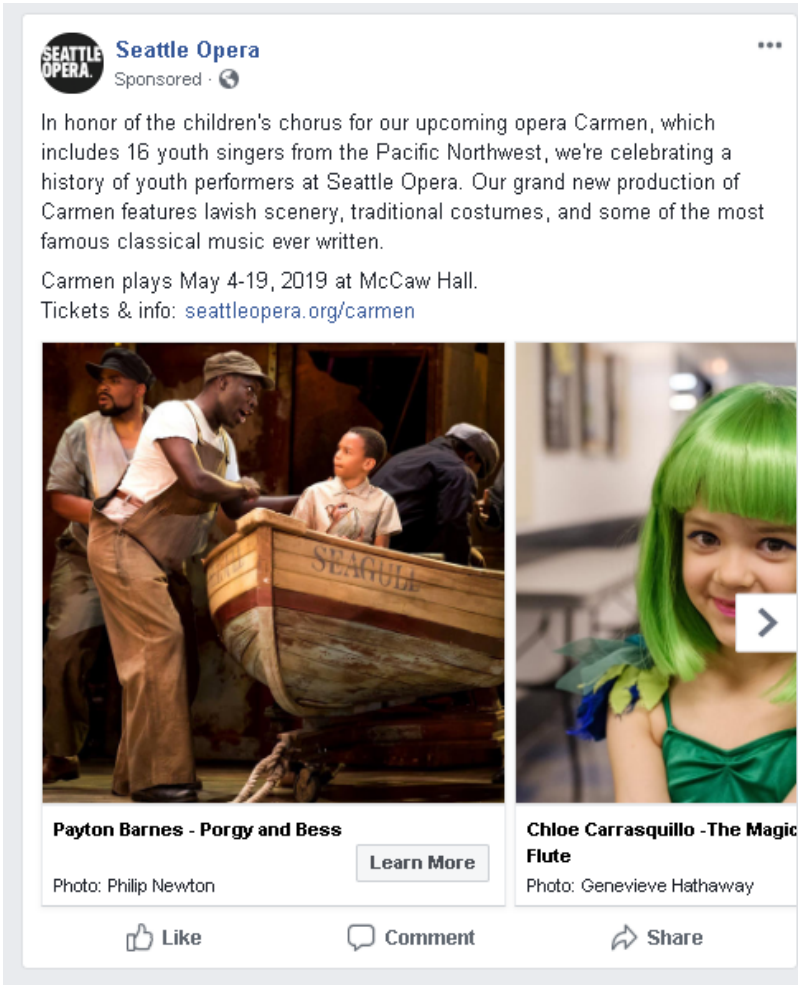


Exploit

My ad trajectory

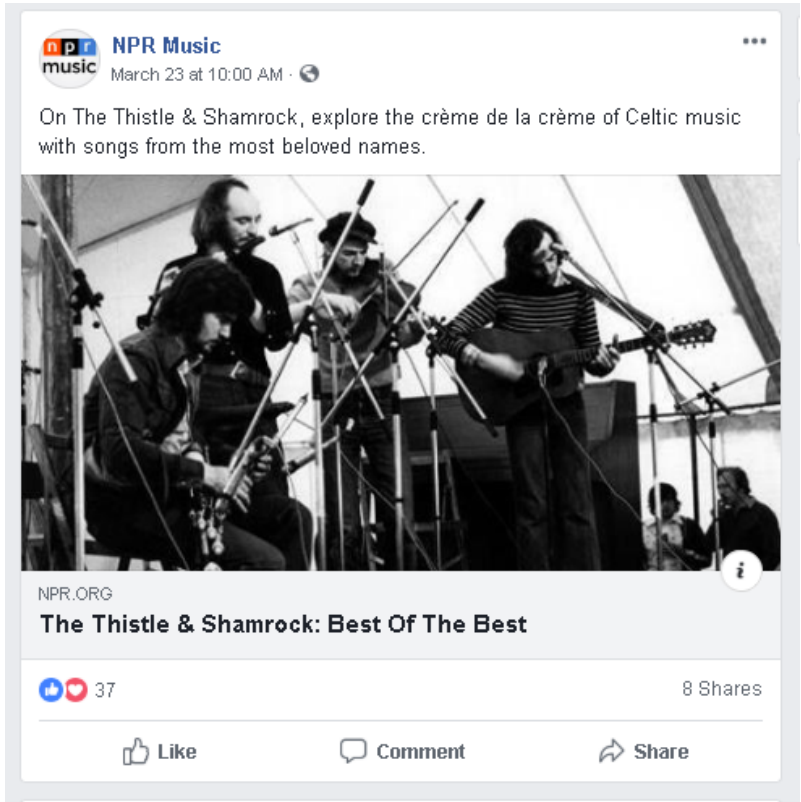
Exploit

My ad trajectory



Explore

My ad trajectory



Exploit

My ad trajectory

Explore

My ad trajectory

Exploit

Upper confidence bound (UCB)

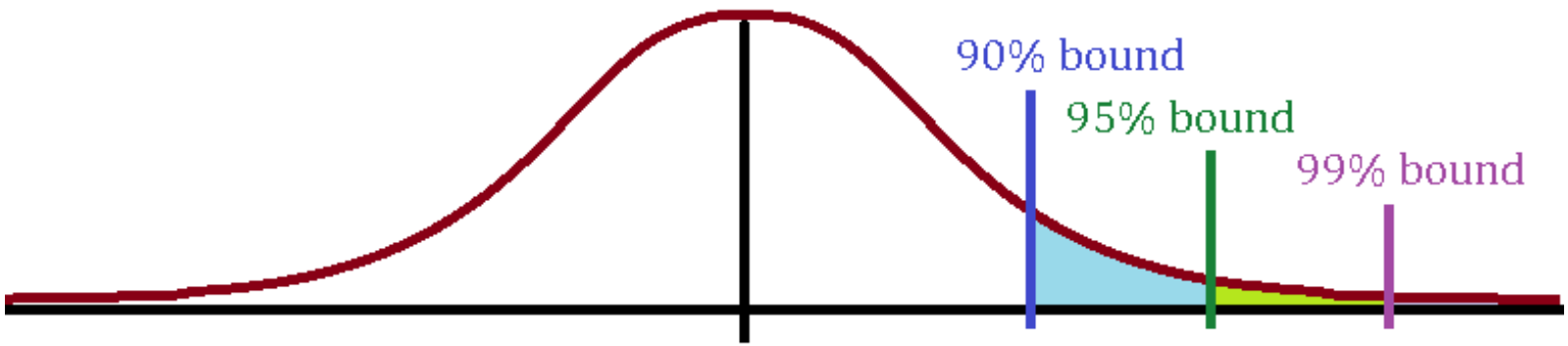
- Consider X a random variable with mean μ and variance σ
- After k observations $x_1, ..., x_k$, the sample mean

$$\bar{x} = \frac{1}{k} \sum_{i=1}^{k} x_i$$

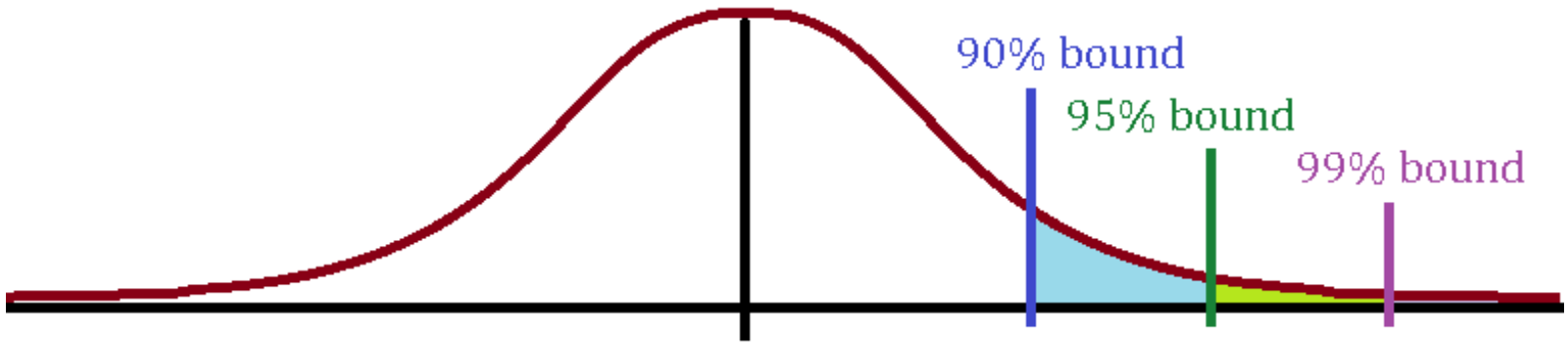
  is also a random variable, with mean $\bar{x}$ and variance σ/n

- The upper confidence bound for 1-α confidence is d where

$$\Pr(\bar{x} > d) \leq \alpha$$

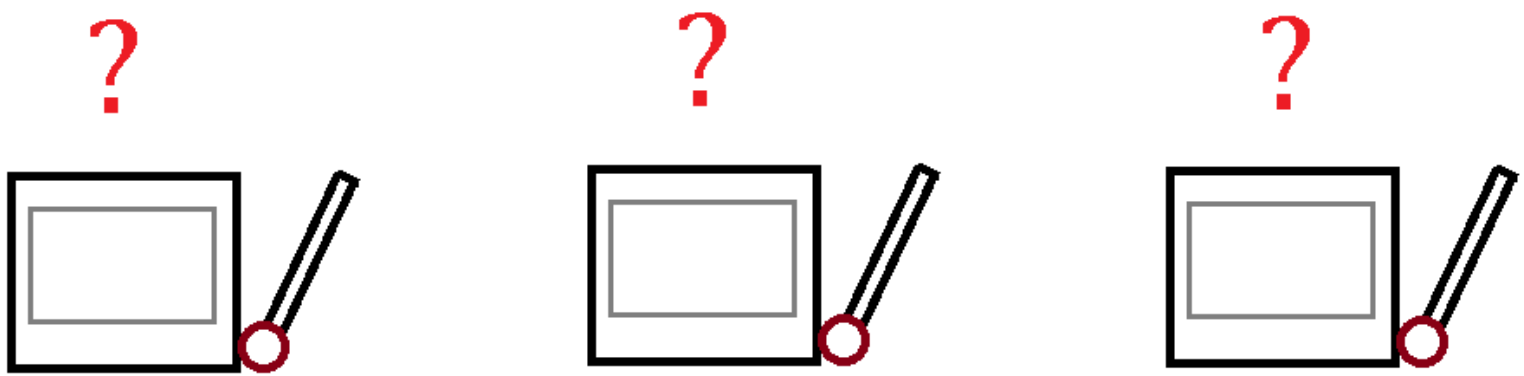


Upper confidence bound method



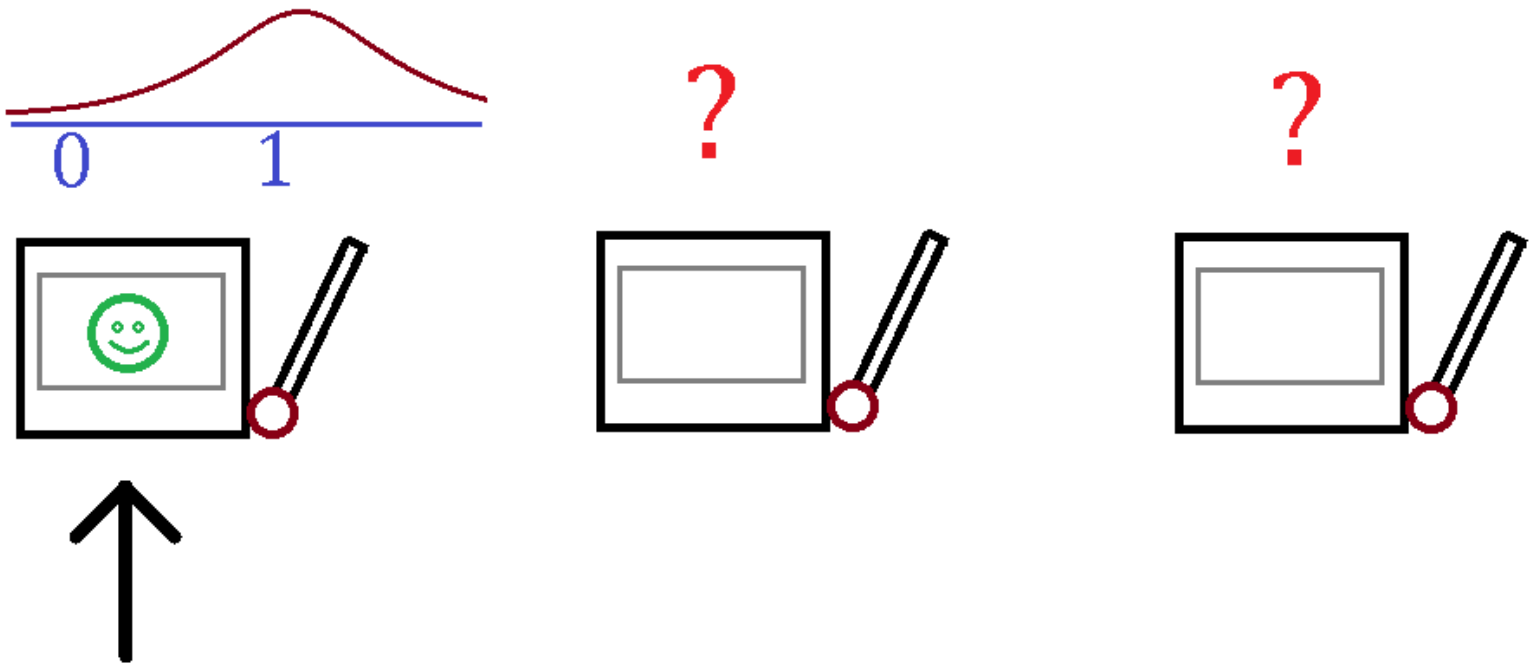
- If you don't know anything, pull any arm
- Now pick the arm with the largest UCB for some confidence $1 - \alpha_t$
- α_t decays to 0
    - never stops exploring, but happens more seldomly for not-best arm
- an "optimistic" method

Upper confidence bound method



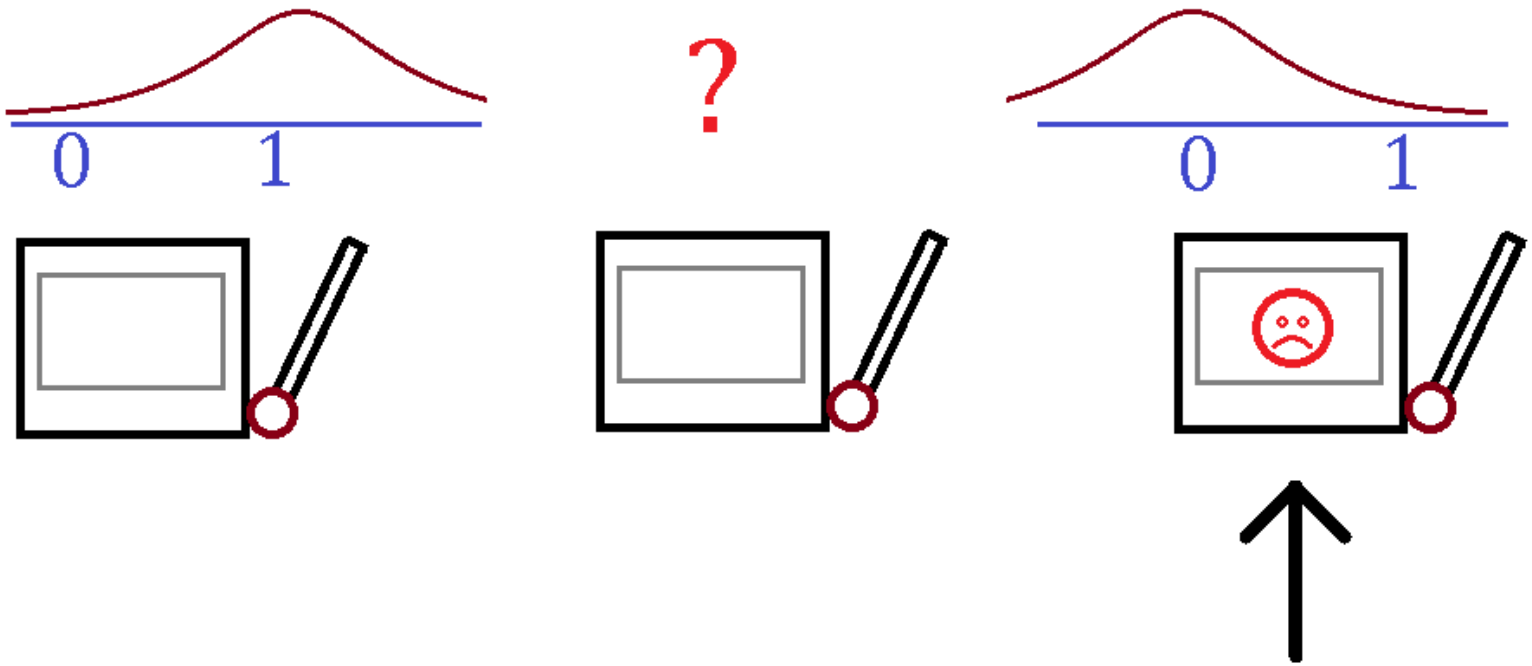
Regret = total pulls - total wins

Upper confidence bound method



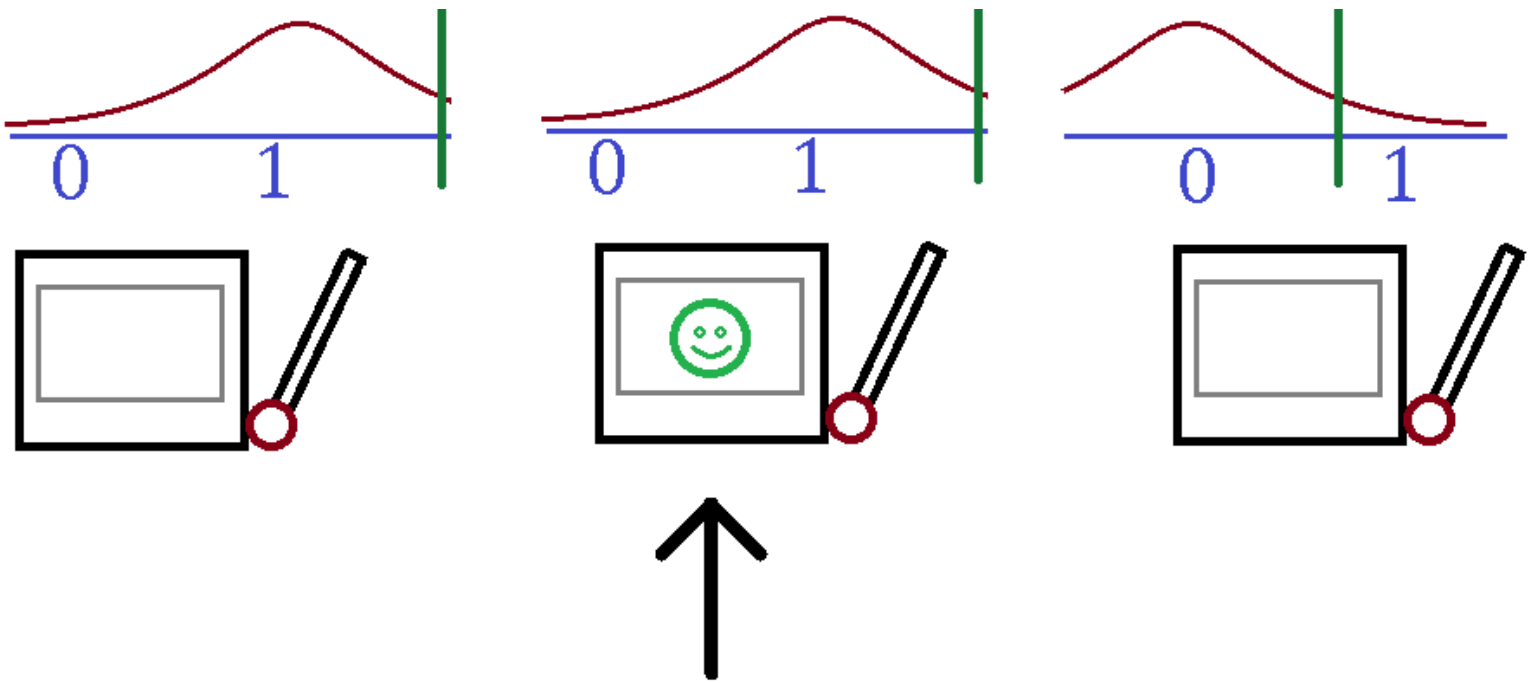
Regret = 1 - 1

Upper confidence bound method



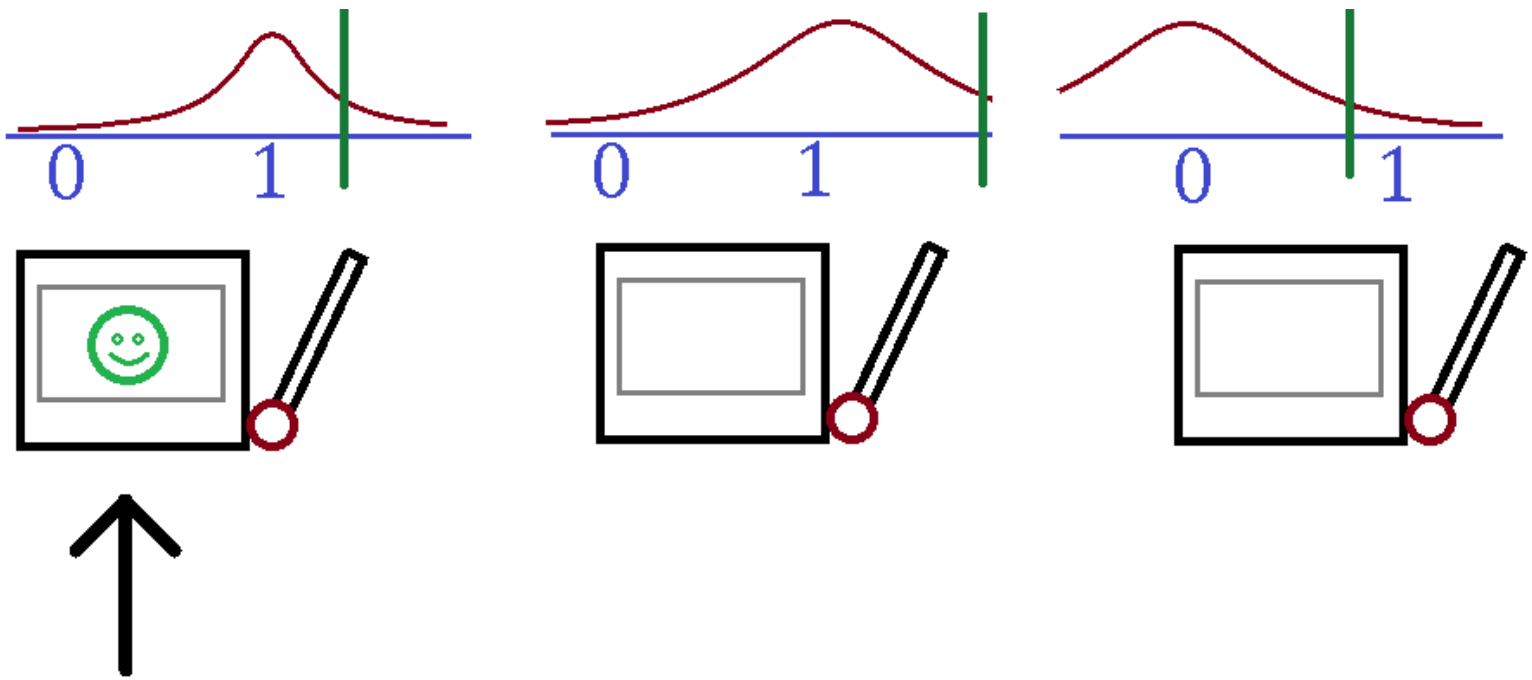
Regret = 2 - 1

Upper confidence bound method



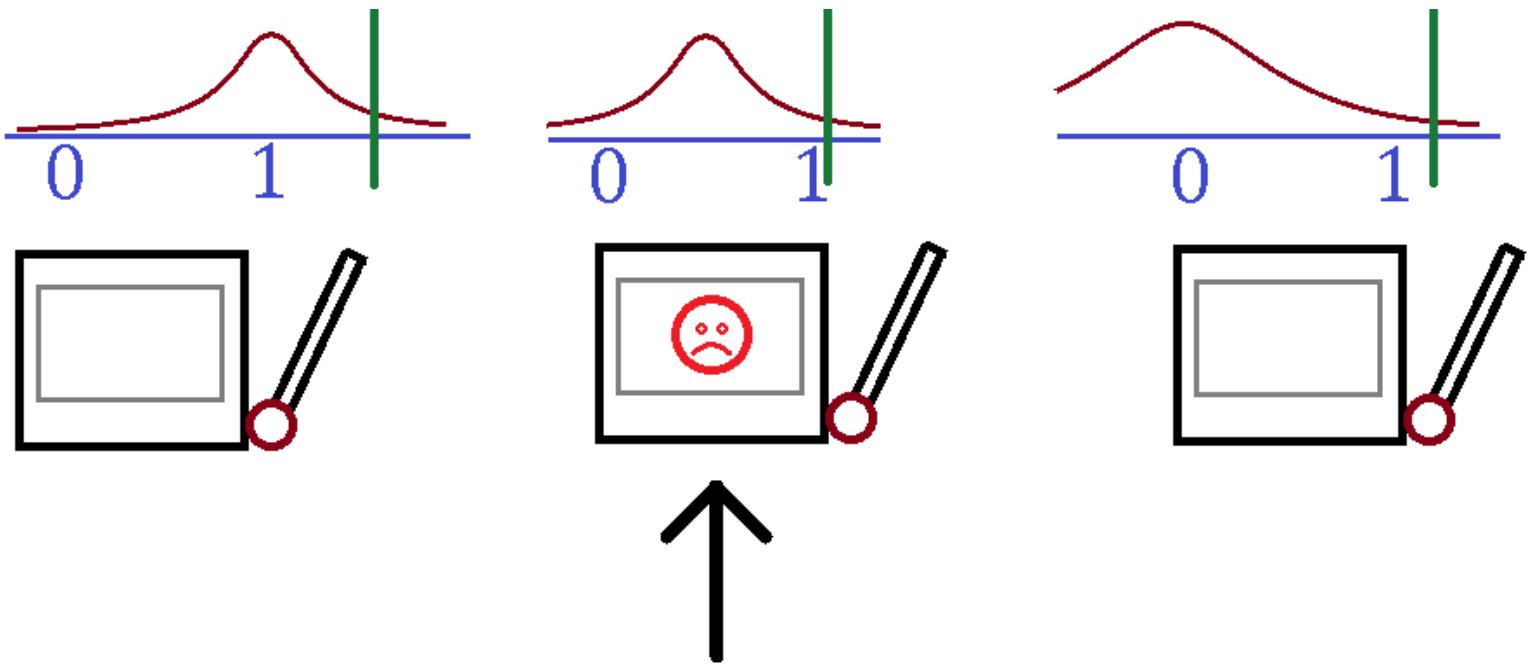
Regret = 3 - 1

Upper confidence bound method



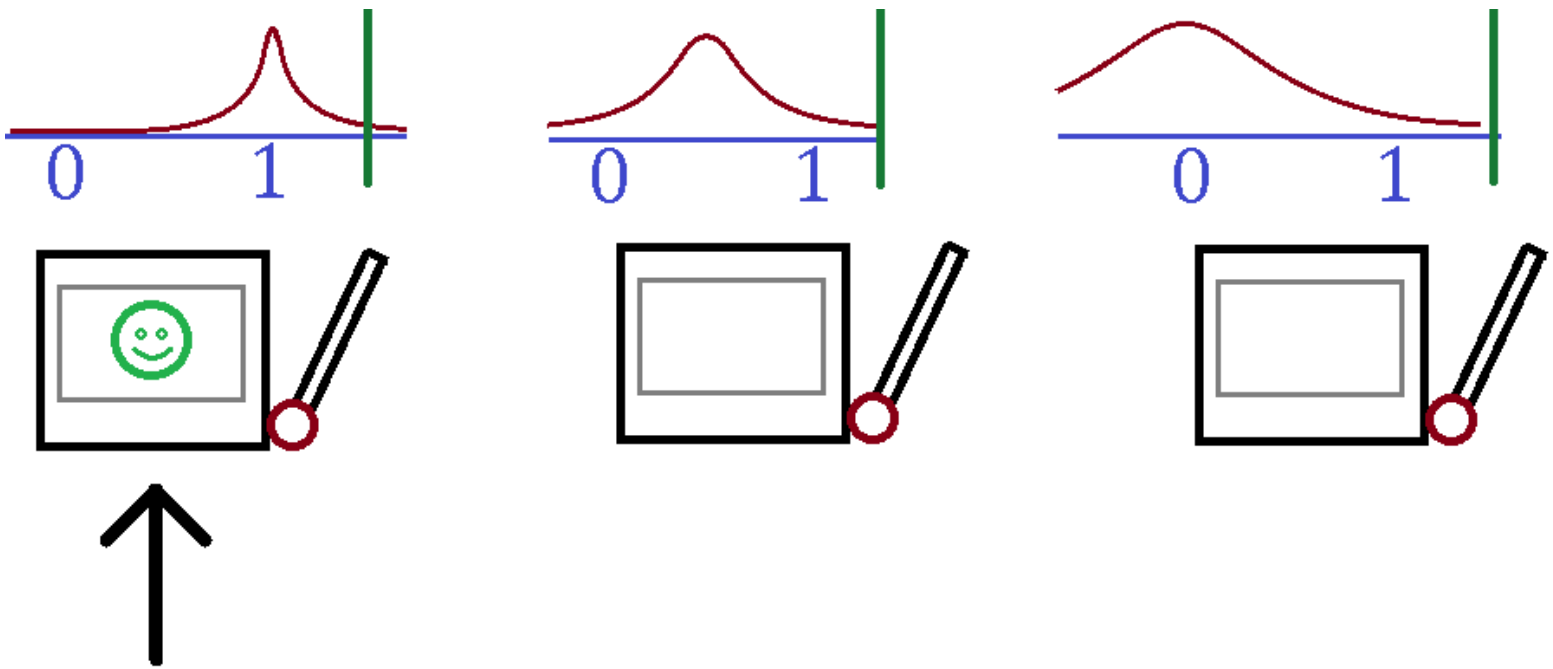
Regret = 4 - 2

Upper confidence bound method



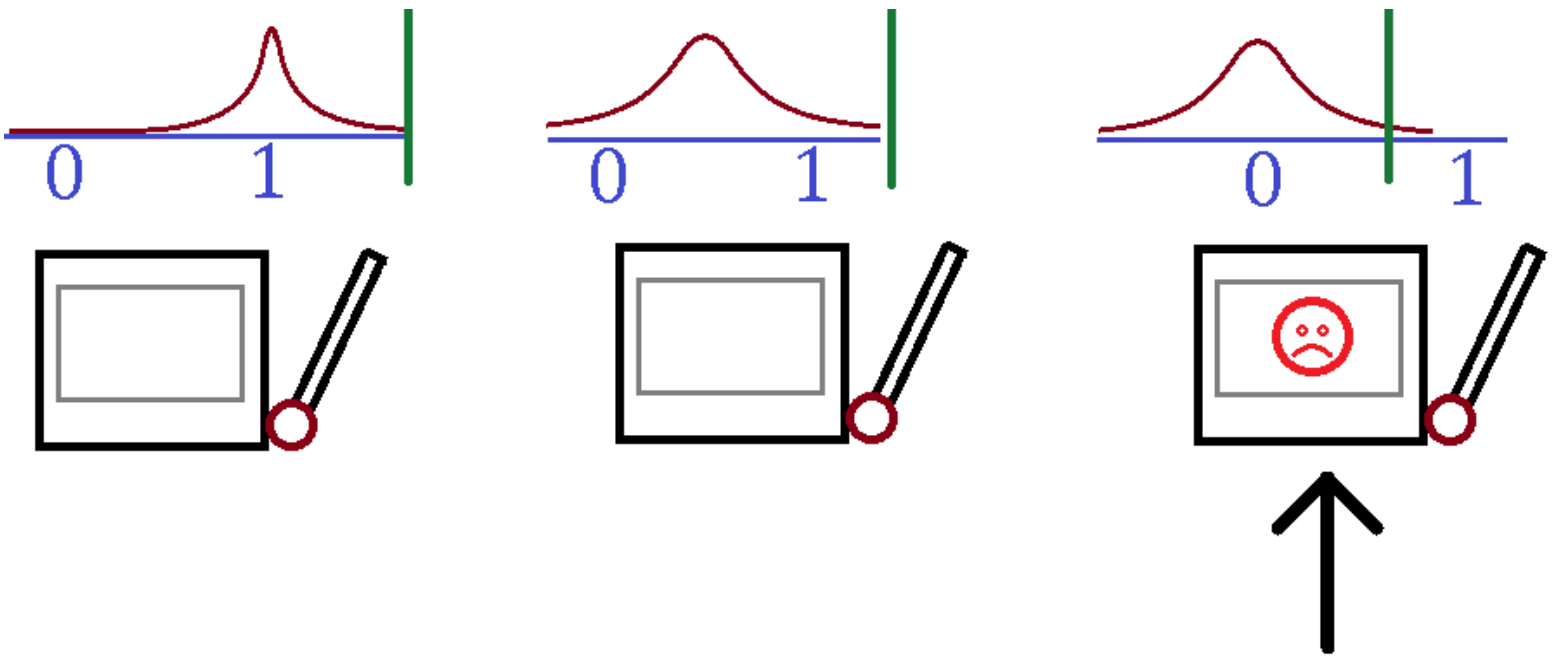
Regret = 5 - 3

Upper confidence bound method



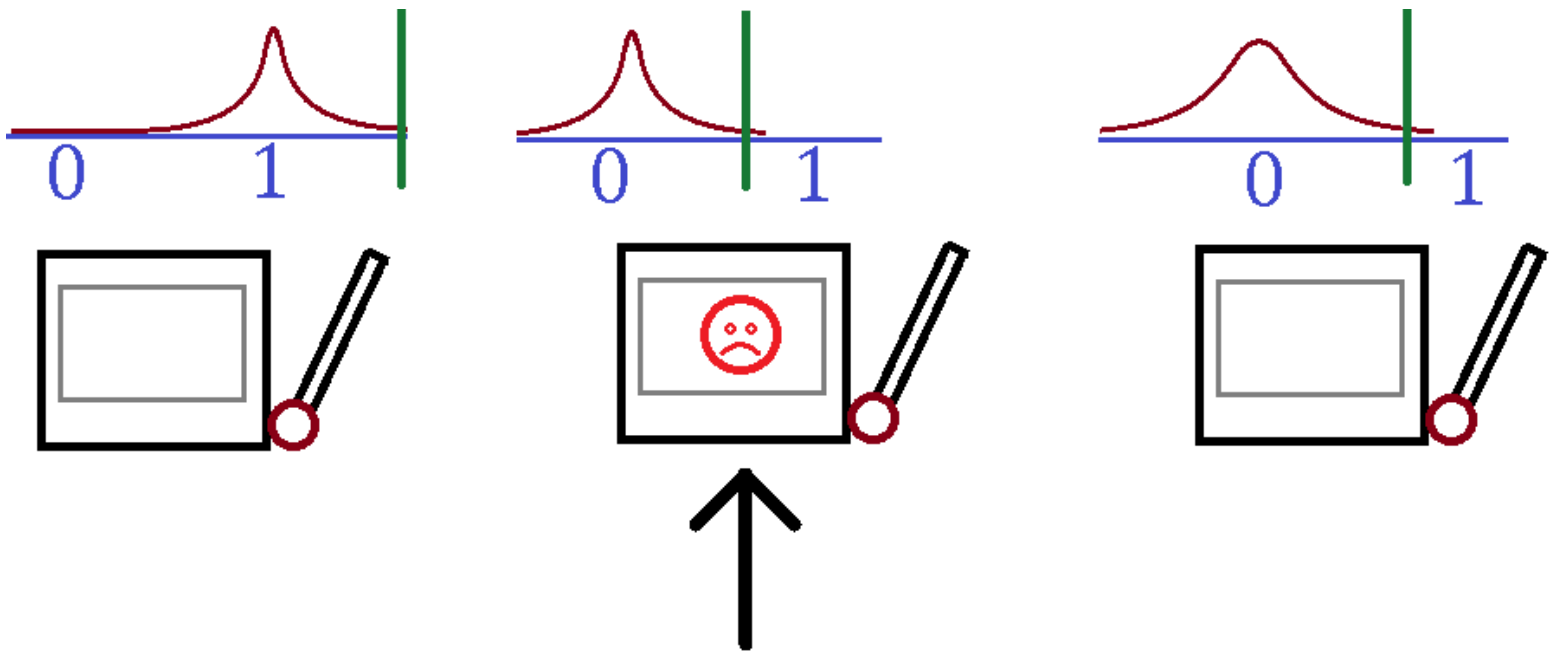
Regret = 6 - 4

Upper confidence bound method



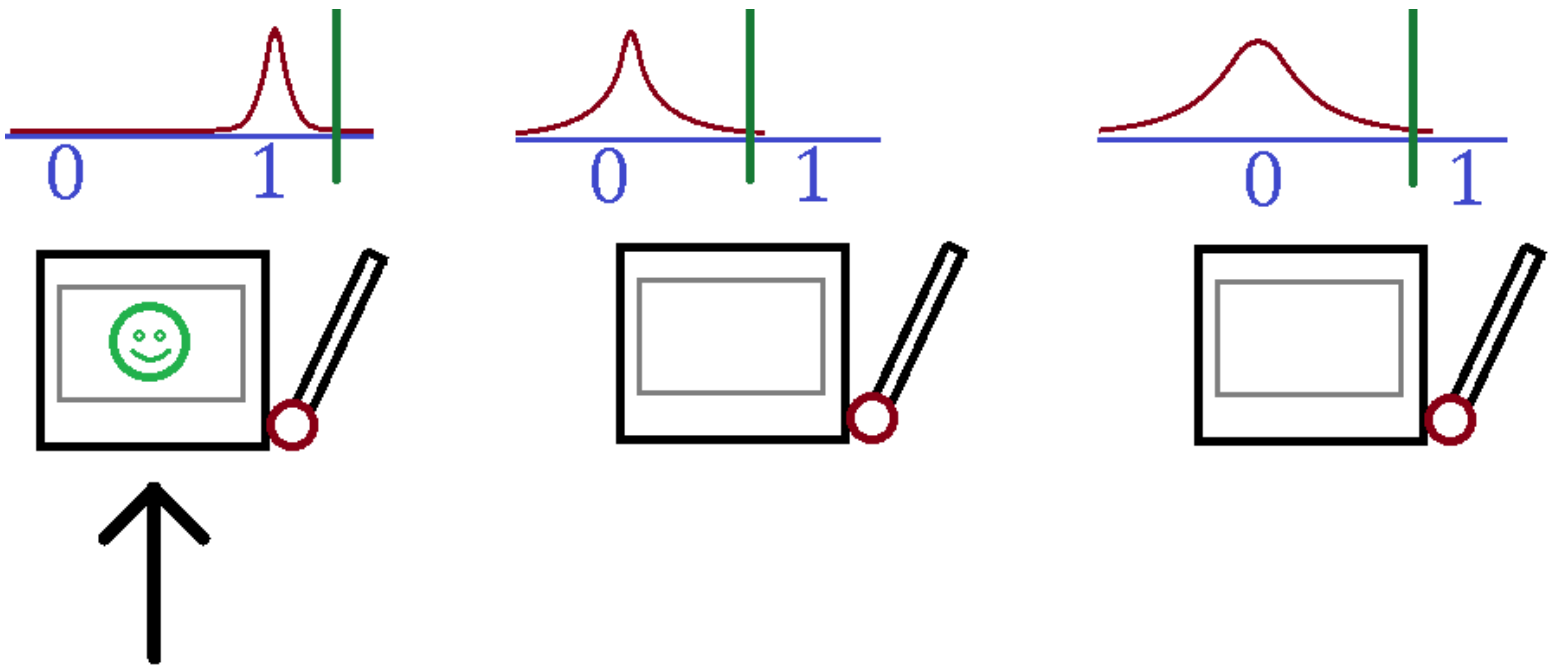
Regret = 7 - 4

Upper confidence bound method



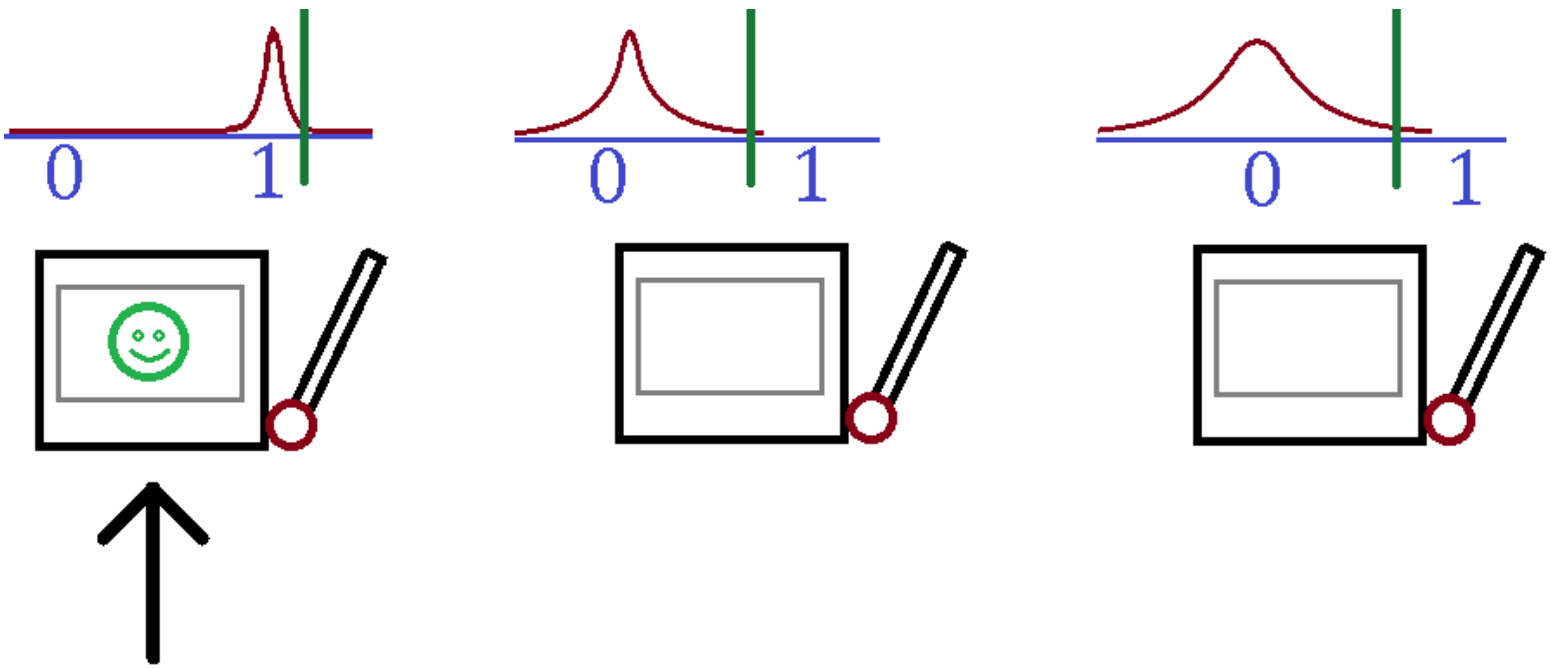
Regret = 8 - 4

Upper confidence bound method



Regret = 9 - 5

Upper confidence bound method

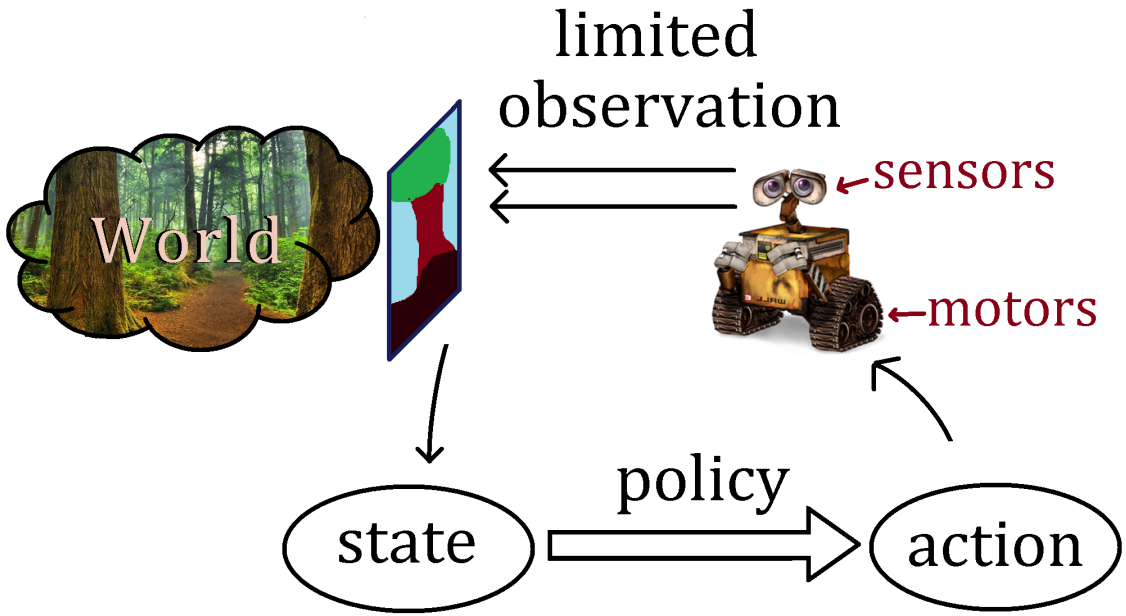


Regret = 10 - 5

Unobserved states

Spot the dog

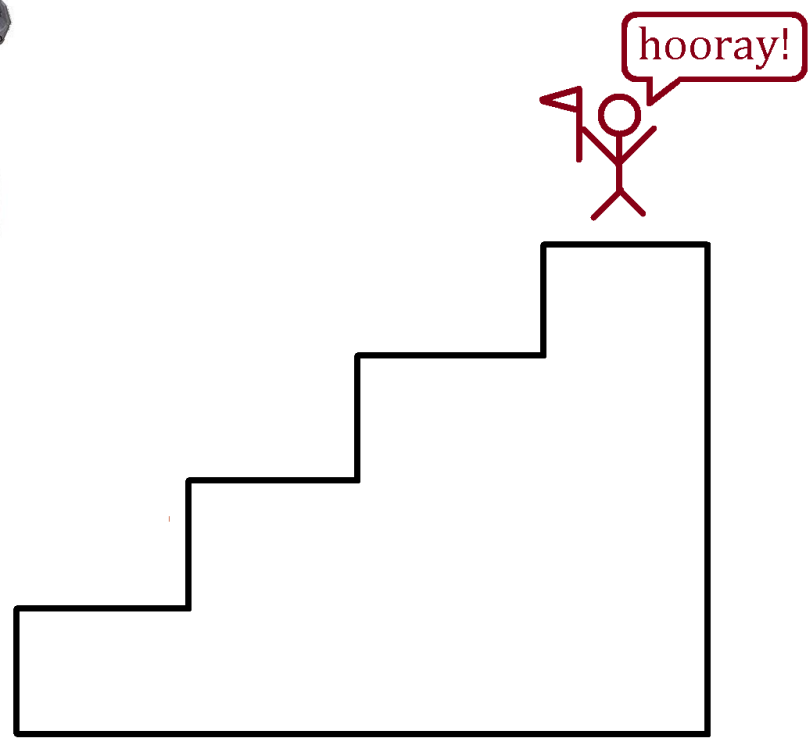
Control cycle



- Robot glimpses world → state updated
- Policy → Robot moves
- Robot gets a new glimpse → state updated
- Rinse and repeat

Robot climbing stairs

POLICY
Move to state of higher reward



What's a good policy?

Goal: Robot climbing the stairs

- Markov state transition:(past given current doesn't matter)

$$\Pr(s^{(t+1)} \mid s^{(t)}, a^{(t)}) \text{ fixed but unknown}$$

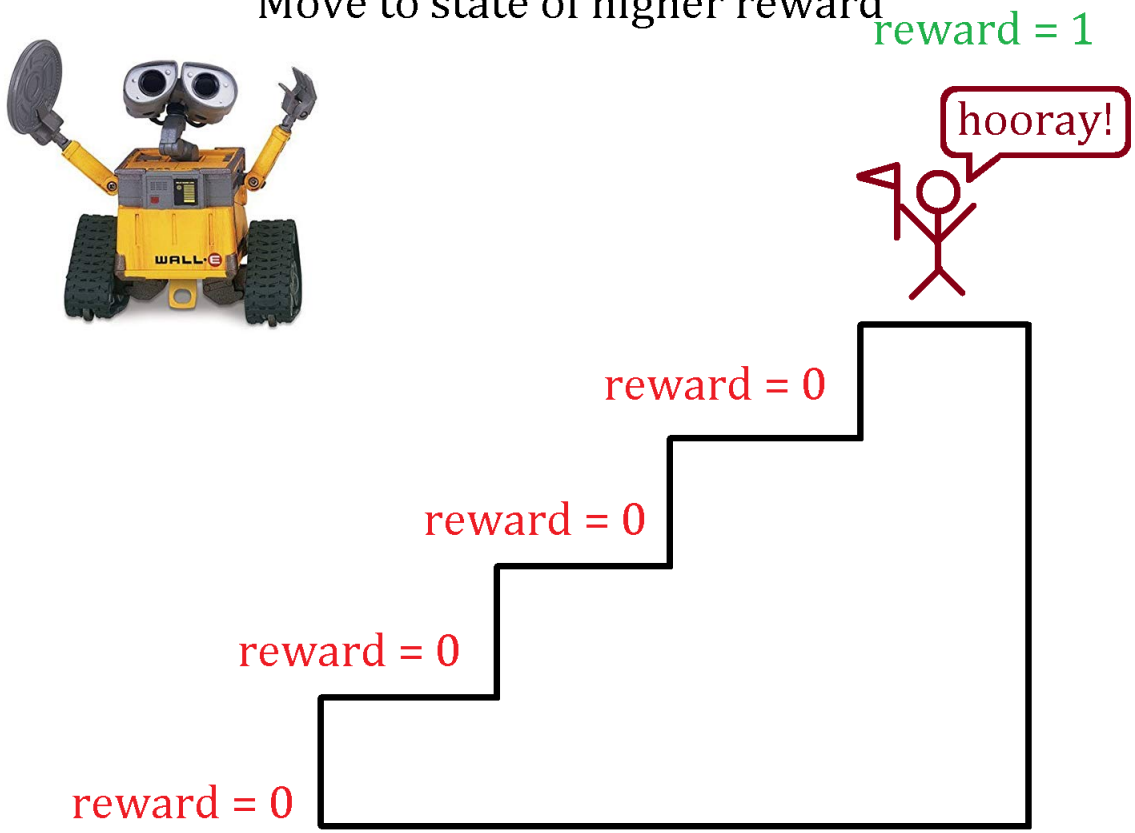
- Reward:

$$r(s) = \begin{cases} 1 & \text{if } s = \text{top step} \\ 0 & \text{otherwise} \end{cases}$$

What's a good policy?

POLICY

Move to state of higher reward



What's a good policy?

Goal: Robot climbing the stairs

- Markov state transition:(past given current doesn't matter)

$$\Pr(s^{(t+1)} \mid s^{(t)}, a^{(t)}) \text{ fixed but unknown}$$

- Reward:

$$r(s) = \begin{cases} 1 & \text{if } s = \text{top step} \\ 0 & \text{otherwise} \end{cases}$$

- Policy (first try)

$$\pi(s) = a = \underset{a}{\operatorname{argmax}} \{\mathbb{E}_{s'|s,a}[r(s')]\}$$

- What's wrong with this picture?

What's a good policy?

Goal: Robot climbing the stairs

- Markov state transition:(past given current doesn't matter)

$$\Pr(s^{(t+1)} \mid s^{(t)}, a^{(t)}) \text{ fixed but unknown}$$

- Reward:

$$r(s) = \begin{cases} 1 & \text{if } s = \text{top step} \\ 0 & \text{otherwise} \end{cases}$$

- Policy (first try)

$$\pi(s) = a = \operatorname*{argmax}_a \{\mathbb{E}_{s'|s,a}[r(s')]\}$$

- What's wrong with this picture? Ans: reward is too sparse!

What's a good policy?

Goal: Robot climbing the stairs

- Markov state transition:

$$s^{(t+1)} = \Pr(s^{(t+1)} \mid s^{(t)}, a^{(t)})$$

- Reward:

$$r(s) = \begin{cases} 1 & \text{if } s = \text{top step} \\ 0 & \text{otherwise} \end{cases}$$

- Value given policy: (discounted reward)

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(s^{(t)}) : s^{(0)} = s, s^{(t)} = \pi(s^{(t-1)})\right], \quad 0 < \gamma < 1$$

- Interpretation: How well will I do in the near future, if I follow this policy?

What's a good policy?

POLICY
Move to state of higher value

value = 1
reward = 1

value = 0.75
reward = 0

value = 0.5
reward = 0

value = 0.25
reward = 0

value = 0
reward = 0

What's a good policy?

- Value given policy: (discounted reward)

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t r(s^{(t)}) : s^{(0)} = s, s^{(t)} = \pi(s^{(t-1)})\right], \quad 0 < \gamma < 1$$

- Optimal policy

$$\pi^*(s) = \operatorname*{argmax}_{\pi} V^{\pi}(s)$$

- "Best" value function

$$V^*(s) = V^{\pi^*}$$

- Chicken and egg problem?

Bellman condition

- Bellman optimality equation: $V^*(s)$ uniquely solves

$$V^*(s) = r(s) + \gamma \max_a \sum_{s'} \Pr(s' \mid s, a) V^*(s')$$

- Optimal policy

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} \Pr(s' \mid s, a) V^*(s')$$

Computing optimal policy and value function

Value iteration

$$\begin{aligned} V^{(0)}(s) &= 0 \; \forall s \\ V^{(t+1)}(s) &= R(s) + \gamma \max_a \sum_{s'} \Pr(s' \mid s, a) \; \forall s \end{aligned}$$

Then $V^{(t)} \overset{t \to \infty}{\to} = V^*$

Policy iteration

$$\begin{aligned} \pi^{(0)}(s) &= 0 \; \forall s \\ V^{(t)}(s) &= R(s) + \gamma \sum_{s'} \Pr(s' \mid s, \pi(s)) V^{(t)}(s) \text{through roll-out} \\ \pi^{(t+1)}(s) &= \operatorname*{argmax}_a \sum_{s'} \Pr(s' \mid s, a) \; \forall s \end{aligned}$$

Then $\pi^{(t)} \overset{t \to \infty}{\to} = \pi^*$

Linear programming

V^* is the solution to the following infinite-dimensional linear program

$$\begin{array}{lll} (P) & \underset{V}{\text{minimize}} & \sum_s V(s) \\ & \text{subject to} & V(s) \geq R(s) + \gamma \sum_{s'} \Pr(s' \mid s, a) V(s') \; \forall a \end{array}$$

Why does this work?

- Feasibility implies

$$V(s) \geq R(s) + \gamma \max_a \sum_{s'} \Pr(s' \mid s, a) V(s')$$

- Optimality implies tightness

$$V(s) = R(s) + \gamma \max_a \sum_{s'} \Pr(s' \mid s, a) V(s')$$

- This means Bellman's condition is satisfied

Dual linear programming

V^* is the solution to the following infinite-dimensional linear program

$$
\begin{aligned}
(D) \qquad & \underset{\mu}{\text{maximize}} && \sum_s \sum_a R(s)\mu(s,a) \\
& \text{subject to} && \sum_a \mu(s',a) = e + \gamma \sum_s \sum_a \Pr(s' \mid s,a)\mu(s,a) \;\forall s' \\
& && \mu(s,a) \geq 0 \;\forall s,a
\end{aligned}
$$

Interpretation

- μ is discounted probability

$$\mu(s,a) = \sum_{t=0}^{\infty} \gamma^t \Pr(s^{(t)} = s, a^{(t)} = a)$$

- Optimal policy

$$\pi^*(s) = \max_a \mu(s,a)$$

Caveats

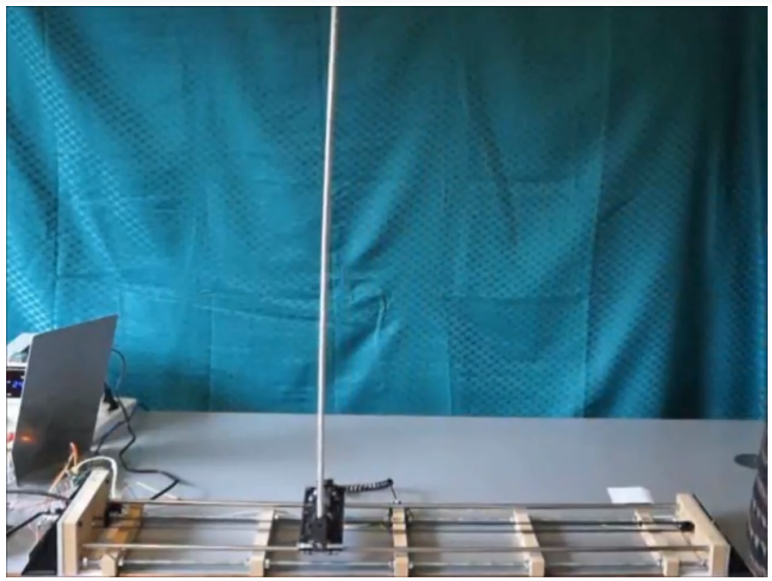
- In general, you cannot really optimize over infinite spaces
  - Discretize state space into grid
  - Discretize functions with basis functions

$$V(s) = \sum_{i=1}^{n} \alpha_i b_i(s)$$

    b_i is fixed, learn α_i

- Even if your state space was finite, a robot can only see its current state
  - Q-learning, SARSA, ...
  - At each iteration, $V(s)$ is only updated for one value of s
  - Alternatively, roll-out (check all s by playing game many times)

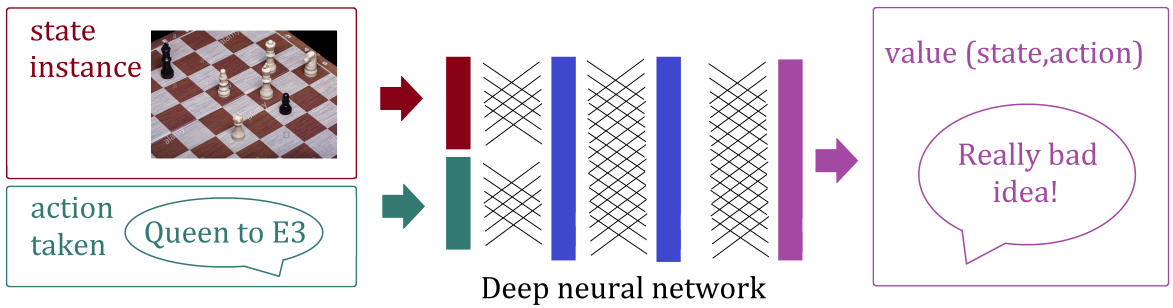
Cart Pole using temporal difference learning



https://www.youtube.com/watch?v=5Q14EjnOJZc

Deep reinforcement learning: Actor-critic

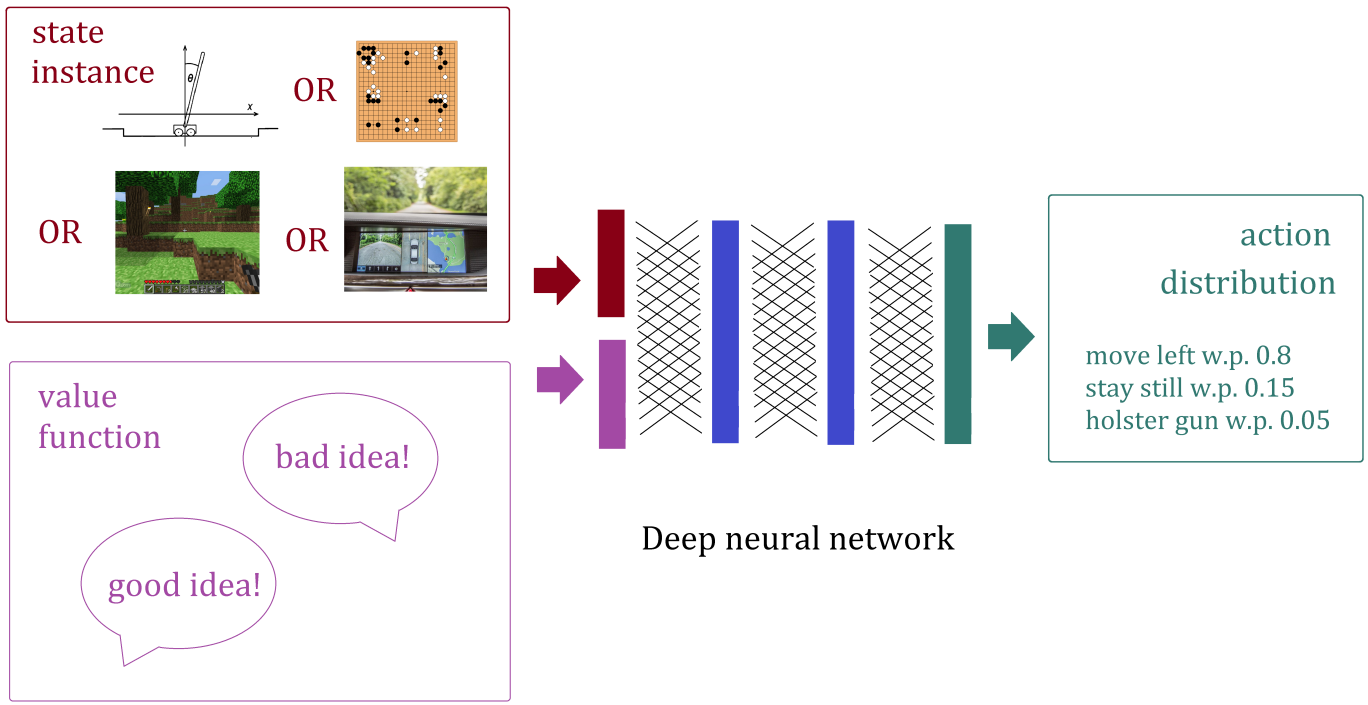
Critic



- Determines the value function given state and action
  - Also called Q function
- Function is approximated by a deep learning network
- This is a good idea when states are represented by images

Deep reinforcement learning: Actor-critic

Actor

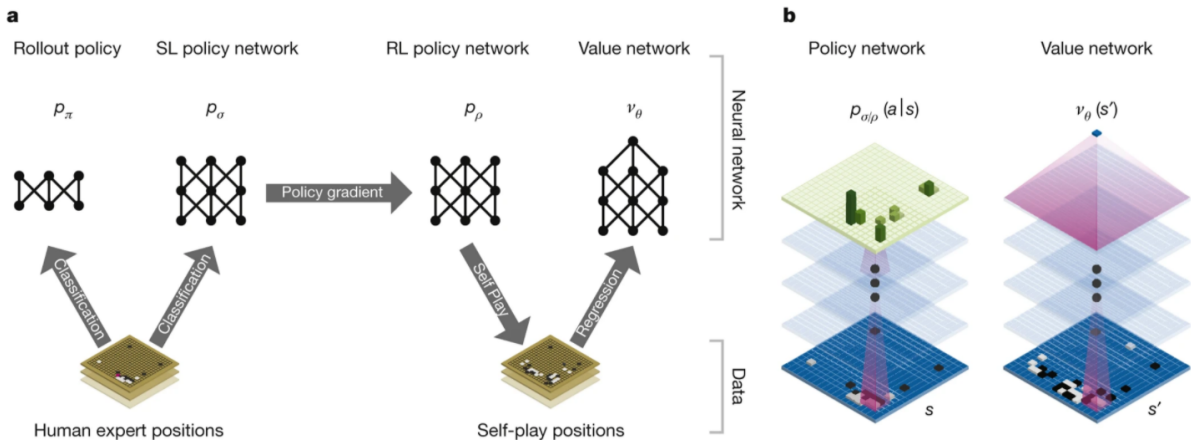


- Determines action probability distribution, given state and value function
- Successes: AlphaGo, Atari, Doom, ...

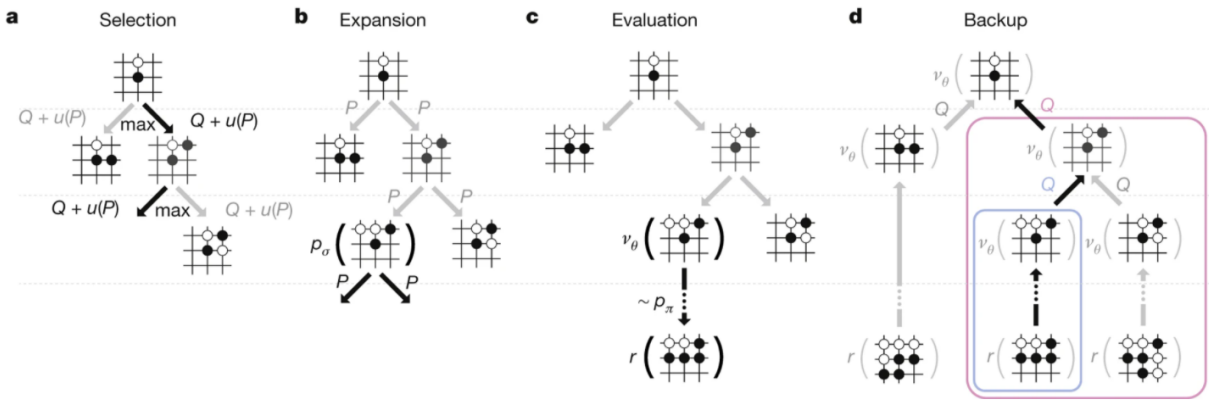
Google's AlphaGo AI wins three-match series against the world's best Go player

Google's AlphaGo AI has once again made the case that machines are now smarter than man — when it comes to games of strategy, at least.

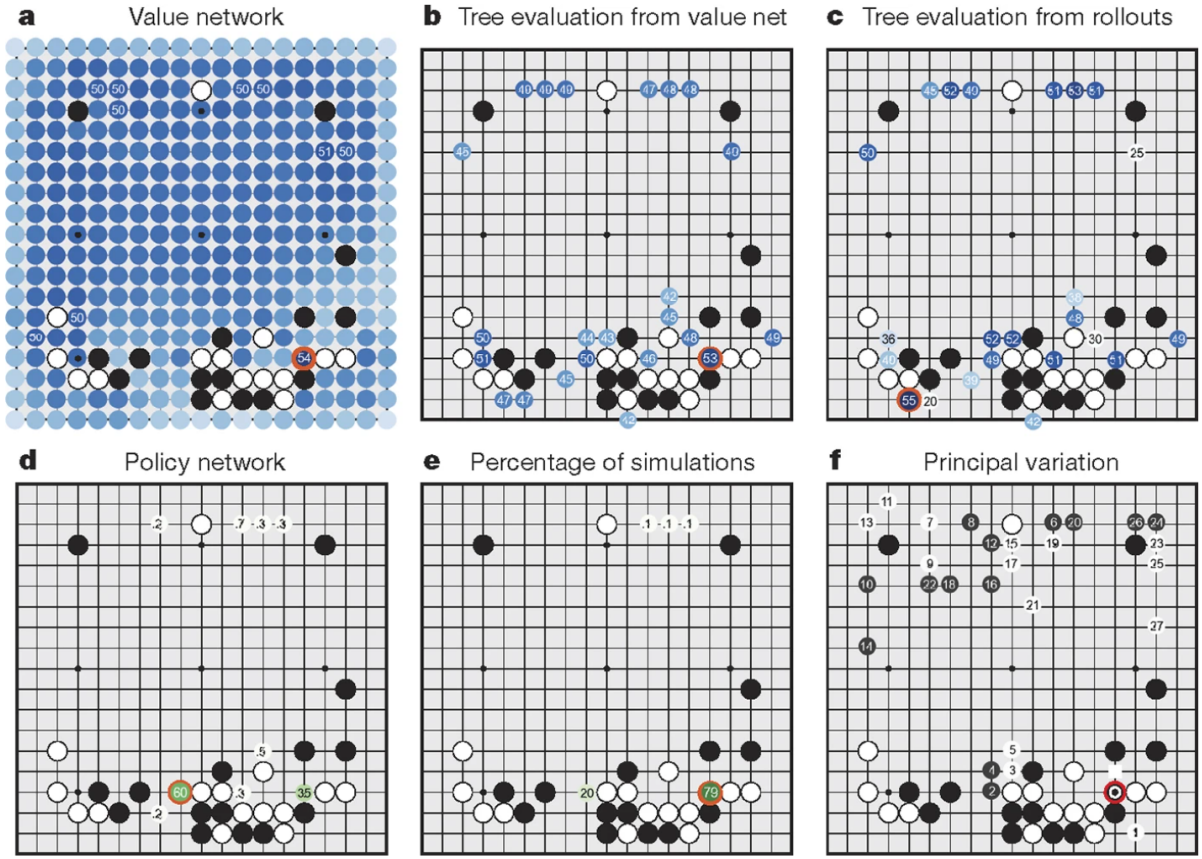
All games of perfect information have an optimal value function, $v^*(s)$, which determines the outcome of the game, from every board position or state s, under perfect play by all players. These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where b is the game's breadth (number of legal moves per position) and d is its depth (game length). In large games, such as chess ($b \approx 35$, $d \approx 80$)[1] and especially Go ($b \approx 250$, $d \approx 150$)[1], exhaustive search is infeasible[2,3], but the effective search space can be reduced by two general principles. First, the depth of the search may be reduced by position evaluation: truncating the search tree at state s and replacing the subtree below s by an approximate value function $v(s) \approx v^*(s)$ that predicts the outcome from state s. This approach has led to superhuman performance in chess[4], checkers[5] and othello[6], but it was believed to be intractable in Go due to the complexity of the game[7]. Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$ that is a probability distribution over possible moves a in position s. For example, Monte Carlo rollouts[8] search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p. Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon[8] and Scrabble[9], and weak amateur level play in Go[10].



a, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set. **b**, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.



a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.



For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a**, Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b**, Action values $Q(s, a)$ for each edge (s, a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c**, Action values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$). **d**, Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e**, Percentage frequency with which actions were selected from the root during simulations. **f**, The principal variation (path with maximum visit count) from AlphaGo's search tree. The moves are presented in a numbered sequence. AlphaGo selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (labelled 1) predicted by AlphaGo.

Configuration and strength[60]

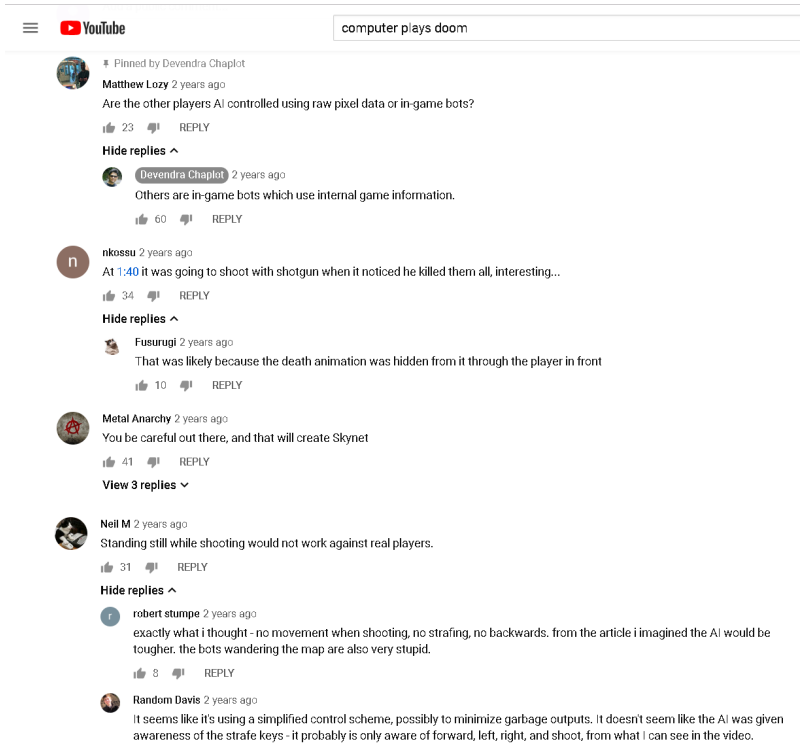
Versions	Hardware	Elo rating	Date	Results
AlphaGo Fan	176 GPUs,[51] distributed	3,144[50]	Oct 2015	5:0 against Fan Hui
AlphaGo Lee	48 TPUs,[51] distributed	3,739[50]	Mar 2016	4:1 against Lee Sedol
AlphaGo Master	4 TPUs,[51] single machine	4,858[50]	May 2017	60:0 against professional players; Future of Go Summit
AlphaGo Zero (40 block)	4 TPUs,[51] single machine	5,185[50]	Oct 2017	100:0 against AlphaGo Lee 89:11 against AlphaGo Master
AlphaZero (20 block)	4 TPUs, single machine	5,018 [61]	Dec 2017	60:40 against AlphaGo Zero (20 block)

https://en.wikipedia.org/wiki/AlphaGo

Computer plays doom

https://www.youtube.com/watch?v=oo0TraGu6QY

Computer plays doom



https://www.youtube.com/watch?v=oo0TraGu6QY

Some more references

- Online convex optimization and no-regret learning: algorithms, guarantees, and applications by Belmega et al.
- Blogs:
  - First part: google stochastic bandits, ϵ-greedy, sublinear regret learning
  - Second part: google Reinforcement learning, Q-learning, actor-critic, deep reinforcement learning
- LP formulation of value / policy optimization:
  - Class slides: http://www.cs.cmu.edu/afs/cs/academic/class/15780-s16/www/slides/mdps.pdf