Reinforcement Learning Lecture 8a:

Multi-armed Bandits
[SutBar] Sec. 2.1-2.7,
[Sze] Sec. 4.2.1-4.2.2

Outline

- Exploration/exploitation tradeoff
- Regret
- Multi-armed bandits
  - ϵ-greedy strategies
  - Upper confidence bounds

Exploration/Exploitation Tradeoff

- Fundamental problem of RL due to the active nature of the learning process

- Consider one-state RL problems known as bandits

Stochastic Bandits

- Formal definition:
  - Single state: S = {s}
  - A: set of actions (also known as arms)
  - Space of rewards (often re-scaled to be [0,1])
- No transition function to be learned since there is a single state
- We simply need to learn the **stochastic** reward function

Origin

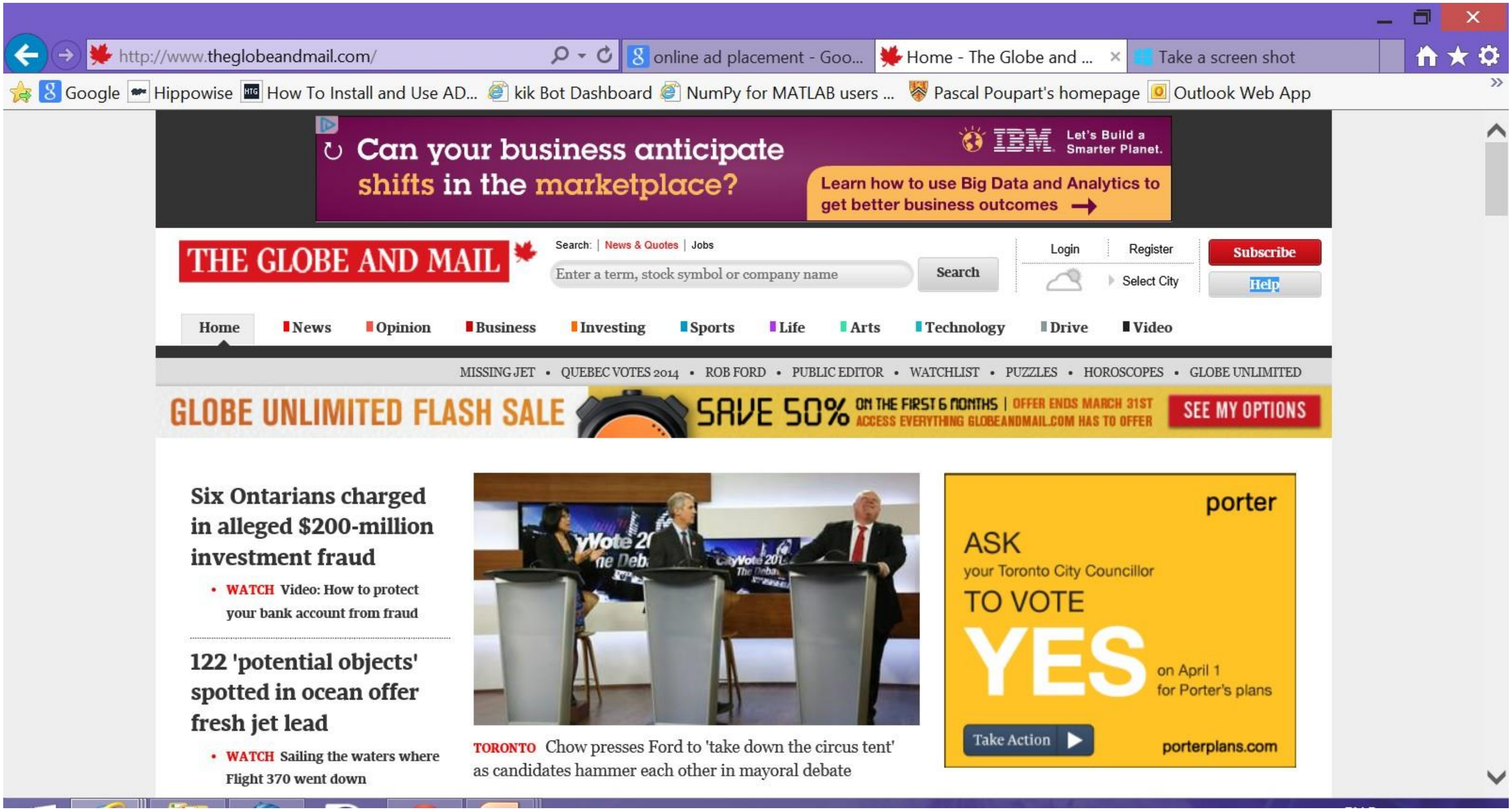
- The term bandit comes from gambling where slot machines can be thought as one-armed bandits.

- Problem: which slot machine should we play at each turn when their payoffs are not necessarily the same and initially unknown?

Examples

- Design of experiments (Clinical Trials)
- Online ad placement
- Web page personalization
- Games
- Networks (packet routing)

Online Ad Optimization



Online Ad Optimization

- Problem: which ad should be presented?
- Answer: present ad with highest payoff

$$payoff = clickThroughRate \times payment$$

  - Click through rate: probability that user clicks on ad
  - Payment: $$ paid by advertiser
    - Amount determined by an auction

Simplified Problem

- Assume payment is 1 unit for all ads
- Need to estimate click through rate

- Formulate as a bandit problem:
  – Arms: the set of possible ads
  – Rewards: 0 (no click) or 1 (click)

- In what order should ads be presented to maximize revenue?
  – How should we balance exploitation and exploration?

Simple yet difficult problem

- Simple: description of the problem is short
- Difficult: no known tractable optimal solution

Simple heuristics

- Greedy strategy: select the arm with the highest average so far
  - May get stuck due to lack of exploration
- ϵ-greedy: select an arm at random with probability ϵ and otherwise do a greedy selection
  - Convergence rate depends on choice of ϵ

Regret

- Let $R(a)$ be the unknown average reward of a
- Let $r^* = \max_a R(a)$ and $a^* = \operatorname{argmax}_a R(a)$

- Denote by $loss(a)$ the expected regret of a

$$loss(a) = r^* - R(a)$$

- Denote by $Loss_n$ the expected cumulative regret for n time steps

$$Loss_n = \sum_{t=1}^{n} loss(a_t)$$

Theoretical Guarantees

- When ϵ is constant, then
  - For large enough t : $\Pr(a_t \neq a^*) \approx \epsilon$
  - Expected cumulative regret: $Loss_n \approx \sum_{t=1}^{n} \epsilon = O(n)$
    - Linear regret

- When $\epsilon_t \propto 1/t$
  - For large enough t: $\Pr(a_t \neq a^*) \approx \epsilon_t = O(\frac{1}{t})$
  - Expected cumulative regret: $Loss_n \approx \sum_{t=1}^{n} \frac{1}{t} = O(\log n)$
    - Logarithmic regret

Empirical mean

- Problem: how far is the empirical mean $\tilde{R}(a)$ from the true mean $R(a)$?
- If we knew that $\left|R(a) - \tilde{R}(a)\right| \leq bound$
  - Then we would know that $R(a) < \tilde{R}(a) + bound$
  - And we could select the arm with best $\tilde{R}(a) + bound$
- Overtime, additional data will allow us to refine $\tilde{R}(a)$ and compute a tighter $bound$.

Positivism in the Face of Uncertainty

- Suppose that we have an oracle that returns an upper bound $UB_n(a)$ on $R(a)$ for each arm based on n trials of arm a.

- Suppose the upper bound returned by this oracle converges to $R(a)$ in the limit:
  - i.e. $\lim_{n \to \infty} UB_n(a) = R(a)$

- Optimistic algorithm
  - At each step, select $argmax_a UB_n(a)$

Convergence

- Theorem: An optimistic strategy that always selects $argmax_a UB_n(a)$ will converge to a^*

- Proof by contradiction:
  - Suppose that we converge to suboptimal arm a after infinitely many trials.
  - Then $R(a) = UB_\infty(a) \geq UB_\infty(a') = R(a')\ \ \forall a'$
  - But $R(a) \geq R(a')\ \forall a'$ contradicts our assumption that a is suboptimal.

Probabilistic Upper Bound

- Problem: We can't compute an upper bound with certainty since we are sampling

- However we can obtain measures f that are upper bounds most of the time
  - i.e., $\Pr\big(R(a) \leq f(a)\big) \geq 1 - \delta$
  - Example: Hoeffding's inequality

$$\boxed{\Pr\left(R(a) \leq \tilde{R}(a) + \sqrt{\frac{\log\left(\frac{1}{\delta}\right)}{2n_a}}\right) \geq 1 - \delta}$$

where n_a is the number of trials for arm a

Upper Confidence Bound (UCB)

- Set $\delta_n = 1/n^4$ in Hoeffding's bound
- Choose a with highest Hoeffding bound

UCB(h)

$V \leftarrow 0, n \leftarrow 0, n_a \leftarrow 0 \quad \forall a$

Repeat until $n = h$

Execute $argmax_a \tilde{R}(a) + \sqrt{\frac{2 \log n}{n_a}}$

Receive r

$V \leftarrow V + r$

$\tilde{R}(a) \leftarrow \frac{n_a \tilde{R}(a) + r}{n_a + 1}$

$n \leftarrow n + 1, n_a \leftarrow n_a + 1$

Return V

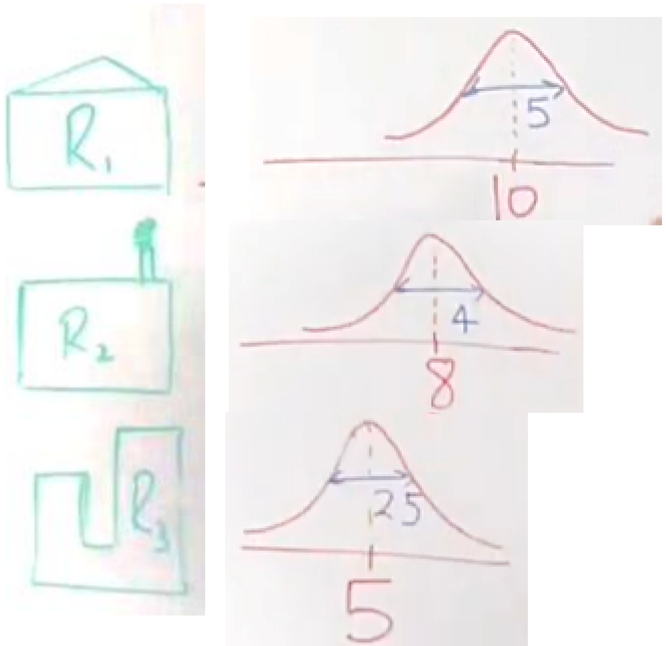
UCB Convergence

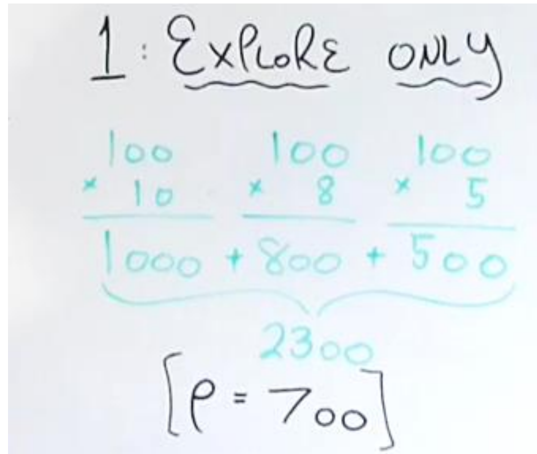
- **Theorem:** Although Hoeffding's bound is probabilistic, UCB converges.

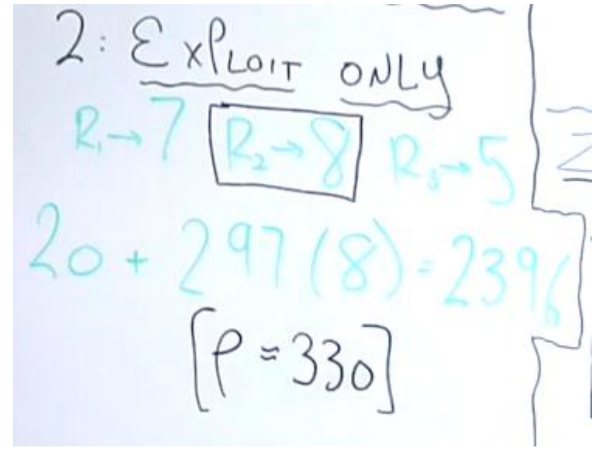
- **Idea:** As n increases, the term $\sqrt{\frac{2\log n}{n_a}}$ increases, ensuring that all arms are tried infinitely often

- Expected cumulative regret: $Loss_n = O(\log n)$
  - Logarithmic regret

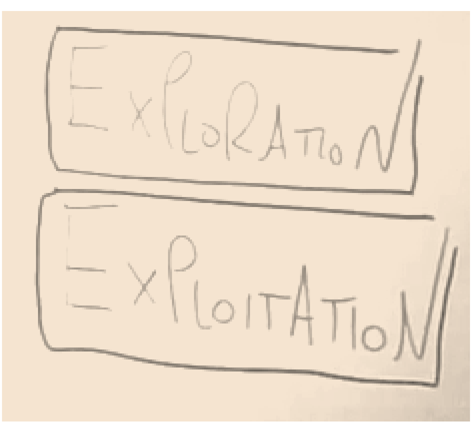
Multi-armed Bandit

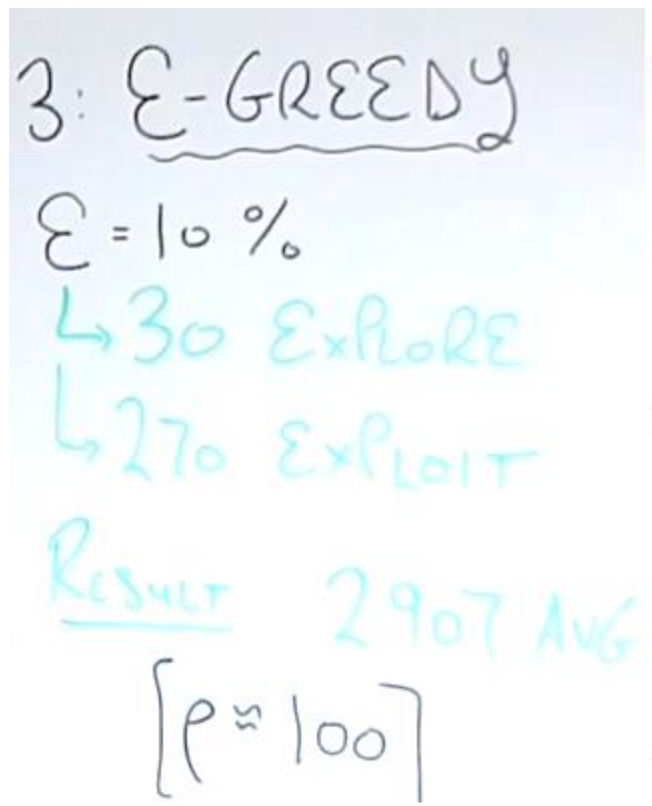






300 Days





Multi-armed Bandit

300 Days

① # RESTAURANTS (n)

$\{R_1, R_2, \ldots, R_n\}$

$\mu_1 \quad \mu_2 \quad \cdots \quad \mu_n$

$= 3 \quad 6 \quad \cdots \quad 3n$

② DEVIATION RATIO (d)

Ex $d = 50\% \Rightarrow (\mu, \frac{\mu}{2})$

$d = 10\% \Rightarrow (\mu, \frac{\mu}{10})$

UCB 1 STRATEGY

AT EACH TIME t, PICK RESTAURANT r, SUCH THAT

$$\left[\hat{\mu}_r + \sqrt{\frac{2 \ln(t)}{N_t(r)}} \right] \text{ IS MAXIMIZED}$$

(HOFFDING'S INEQUALITY)

	d=50% n=3	d=10% n=3	d=50% n=10	d=10% n=10	d=50% n=100	d=10% n=100
Explore	33%	33%	45%	45%	49%	49%
Exploit	11%	0.4% ☆	13%	4%	23% ☆	19% ☆
ε=10%	6%	4%	12%	8%	41%	38%
UCB1	5% ☆	1%	10% ☆	3% ☆	38%	34%

Codes

https://mybinder.org/v2/gh/ritvikmath/Time-Series-Analysis.git/HEAD

In [89]:

```python
import numpy as np
from random import choice
```

In [90]:

```python
class Restaurant:
    def __init__(self, mu, dev):
        self.mu = mu
        self.dev = dev
    def sample(self):
        return np.random.normal(self.mu, self.dev)
```

In [91]:

```python
def explore_only(candidates, num_days):
    scores = []
    for _ in range(num_days):
        scores.append(choice(candidates).sample())
    return sum(scores)
```

In [92]:

```python
def exploit_only(candidates, num_days):
    scores = [c.sample() for c in candidates]
    chosen = candidates[np.argmax(scores)]
    for _ in range(num_days - len(candidates)):
        scores.append(chosen.sample())
    return sum(scores)
```

In [93]:

```python
def epsilon_greedy(candidates, num_days, epsilon=0.05):
    scores = []
    history = {idx: [c.sample()] for idx,c in enumerate(candidates)}
    for _ in range(num_days - len(candidates)):
        p = np.random.random()
        #explore
        if p < epsilon:
            chosen = choice(candidates)
        #exploit
        else:
            chosen = candidates[sorted(history.items(), key=lambda pair: np.mean(pair[1]))[-1][0]]
        score = chosen.sample()
        scores.append(score)
        history[candidates.index(chosen)].append(score)
    return sum(scores)
```

In [94]:

```python
def ucb1(candidates, num_days):
    scores = []
    history = {idx: [c.sample()] for idx,c in enumerate(candidates)}
    for t in range(len(candidates), num_days):
        mu_plus_ucb = [np.mean(history[idx]) + np.sqrt(2*np.log(t) / len(history[idx])) for idx in range(len(candidates))]
        chosen = candidates[np.argmax(mu_plus_ucb)]

        score = chosen.sample()
        scores.append(score)
        history[candidates.index(chosen)].append(score)
    return sum(scores)
```

In [171…

```
dev_factor = 0.5
num_restaurants = 3

mu_vals = [3*i for i in range(1,num_restaurants+1)]
dev_vals = [mu*dev_factor for mu in mu_vals]
mu_dev_pairs = zip(mu_vals, dev_vals)

candidates = [Restaurant(mu,dev) for mu,dev in mu_dev_pairs]

num_days = 300

optimal_average = max(mu_vals)*num_days
```

```
explore_only_vals = []
for _ in range(1000):
    val = explore_only(candidates, num_days)
    explore_only_vals.append(val)
print('Explore Only Mean Regret: %s'%((optimal_average - np.mean(explore_only_vals)) / optimal_average))
```

```
Explore Only Mean Regret: 0.33400345242040025
```

```
exploit_only_vals = []
for _ in range(1000):
    val = exploit_only(candidates, num_days)
    exploit_only_vals.append(val)
print('Exploit Only Mean Regret: %s'%((optimal_average - np.mean(exploit_only_vals)) / optimal_average))
```

```
Exploit Only Mean Regret: 0.10974979914722435
```

```
epsilon_greedy_vals = []
for _ in range(1000):
    val = epsilon_greedy(candidates, num_days, 0.1)
    epsilon_greedy_vals.append(val)
print('Epsilon Greedy Mean Regret (10%%): %s'%((optimal_average - np.mean(epsilon_greedy_vals)) / optimal_average))
```

```
Epsilon Greedy Mean Regret (10%): 0.061901290618584424
```

```
ucb1_vals = []
for _ in range(1000):
    val = ucb1(candidates, num_days)
    ucb1_vals.append(val)
print('UCB1 Mean Regret: %s'%((optimal_average - np.mean(ucb1_vals)) / optimal_average))
```

```
UCB1 Mean Regret: 0.05807450789812113
```

Summary

- Stochastic bandits
  - Exploration/exploitation tradeoff
- ϵ-greedy and UCB
  - Theory: logarithmic expected cumulative regret

- In practice:
  - UCB often performs better than ϵ-greedy
  - Many variants of UCB improve performance